

Xplor8 - Version 1

Manual

This page intentionally left blank

CONTENTS

1	INTRODUCTION	7
1.1	The Program	7
1.2	This Manual	7
1.3	Hardware	7
1.4	Xplor8 Source Code	8
1.5	Philosophy	8
1.5.1	Keyboard Input	8
1.5.2	Default Hexadecimal Number Entry	8
1.5.3	Screen Appearance	8
1.5.4	Command Syntax	8
1.6	Development History	8
1.7	Getting Started	9
1.8	Note on the Author	9
1.9	Origin of the program logo	9
1.10	An Appeal	9
2	MISCELLANEOUS	10
2.1	Installing	10
2.2	Uninstalling	10
2.3	Registry Entries	10
2.4	Project Files	10
2.5	Supporting Files	11
2.6	Remote Reset	11
2.7	Monitor/User Mode Switching	11
2.8	Local and MCU-controlled Memory	11
2.9	USB-to-Serial Adapters	12
3	MON08 MONITOR FIRMWARE	13
3.1	Introduction	13
3.2	Utility Files	14
3.3	Entering Monitor Mode	14
3.4	Security Bytes	14
3.5	Baud Rates	15
4	MEMORY MANAGEMENT	16
4.1	Writing to RAM	16
4.2	Writing to FLASH	16
4.3	Managing the FLBPR Register	16
4.4	Memory Map Display	16
5	DEBUGGING	17
5.1	Breakpoints	17
5.2	SWI in user code	17
6	PROGRAM FEATURES	18
6.1	Screen Layout	18
6.2	Main Menu	19
6.3	Speedbuttons	19

6.4	Output Window	19
6.5	Command History Window	20
6.6	Command Edit Box	20
6.7	Status Line	21
6.8	Information Sidebar	21
6.8.1	CPU Registers Display	21
6.8.2	T/G, L/U and BR addresses	22
6.8.3	Start / Break Point	22
6.8.4	Watch Window	22
6.9	Add to Watch	23
6.10	File Menu	23
6.11	View Menu	24
6.12	Actions Menu	25
6.13	Macro Menu	26
6.14	Settings Menu	26
6.15	Help Menu	26
6.16	Keyboard Shortcuts	27
6.17	Symbol Table/Register Display	28
6.18	Base Converter	28
6.19	Security Bytes Log Viewer	29
7	CONFIGURATION	31
7.1	Settings Dialog	31
7.2	Settings>General	31
7.3	Settings>COM Port	33
7.4	Settings>Macros	35
7.5	Settings>Debug	36
7.6	Settings>Security Bytes	38
7.7	Settings>Notes	39
8	COMMANDS	40
8.1	General information	40
8.2	Path Tokens	41
8.3	Labels instead of Addresses	42
8.4	Clear Local Memory	43
8.5	Clear Output Window	43
8.6	Connect / Disconnect	43
8.7	Edit Flash Byte	44
8.8	Fill Memory	45
8.9	Find Bytes	46
8.10	Find Next	47
8.11	Go (Run)	48
8.12	List Macros	49
8.13	List Memory	50
8.14	Load Memory	51
8.15	Mass Erase	52
8.16	Mode Switch	52
8.17	Modify Memory	53
8.18	Page Erase	54
8.19	Pause	54

8.20	Register Display and Change	55
8.21	Reset	56
8.22	Save Memory	57
8.23	Send Bytes	58
8.24	Launch Terminal Window	59
8.25	Trace	60
8.26	Step Over	61
8.27	Unassemble	62
8.28	Using a Symbol Table	63
8.29	Unlock	65
8.30	Verify	66
8.31	Verify Erase	67
9	AUTOMATION	68
9.1	Introduction	68
9.2	Macros	68
9.3	Boot Script	69
9.4	Autostart Macro	70
9.5	Replaceable Parameters	70
9.6	Playing and Recording Macros	71
9.6.1	Playing	71
9.6.2	Recording	71
9.6.3	Stopping Playing or Recording	72
9.7	Macro Editor	72
10	TERMINAL WINDOW	74
10.1	Introduction	74
10.2	Window Layout and Features	74
10.3	Interaction with the Monitoring/Debugging Functions	75
10.4	Terminal Window Menu Items	75
10.5	Terminal Window Settings	76
10.6	Terminal Demonstration Program	77
11	ERRORS	78
11.1	Communication and Echo Errors	78
11.1.1	Comms Error	78
11.1.2	Echo Error	78
11.2	Diagnostics	79
11.3	Error Report	79
11.4	Command Line Errors	79
	APPENDIX A - ACKNOWLEDGMENTS	80
	APPENDIX B - HARDWARE	81
	APPENDIX C - COMMAND SUMMARY	83
	APPENDIX D - SETUP FOR DIFFERENT MCU's	85
	APPENDIX E - COMMAND-LINE ERROR MESSAGE SUMMARY	86

APPENDIX F - RS232 COMMUNICATIONS	89
APPENDIX G - GETTING STARTED and QUICK TOUR	93
APPENDIX H - DISTRIBUTION FILES	98
APPENDIX J - Xplor8 REVISION HISTORY & KNOWN BUGS	100

<p>Note: All Trade Marks and Trade Names mentioned in this manual are, and remain, the property of their respective owners.</p>

1 INTRODUCTION

1.1 The Program

Xplor8 is a simple loader/de-bugger/monitor interface for the Freescale (Motorola) HC908 micro controller units. I developed it as a simple alternative to some of the all-singing and dancing commercial programs, which I found a bit daunting for a beginner. It is not intended as an integrated development environment, but as a basic GUI interface to the MON08 firmware program built into these chips. It should be compatible with all the Microsoft Windows 32-bit operating systems.

It has been tested with the Windows 98SE and Windows 2000 operating systems and I believe that there should be no difficulty with other Microsoft Win32 OS. I have used it with the MC68HC908QY4, 908JL3, 908JB8 and 908GP32 chips.

The development of Xplor8 follows on from my work on JBug11, a similar interface for the Freescale HC11 MCU's. My aim is to make the program as obvious in its workings as possible and I rely on user feedback as much as anything for improvements.

1.2 This Manual

The Manual is laid out in the following chapters:

- Introduction
- Miscellaneous
- MON08 Monitor Firmware
- Memory Management
- Debugging
- Program Features
- Configuration
- Commands
- Automation
- The Terminal Window
- Errors
- Appendices

1.3 Hardware

Development of this program was carried out with an HC908QY4 micro controller mounted in a purpose-made target board. See [Appendix B](#) for the circuit of this board, and also for a typical hardware layout. It has also been tested with various evaluation boards:

- The HC08 Welcome Kit from Elektronikladen, using the HC908GP32
<http://elmicro.com/de/kit08.html>
- The USB08 Starter Kit from Elektronikladen, using the 908JB8
<http://elmicro.com/de/usb08.html>
- The NITRON EXPRESS Rapid Development PCB from Smith Machine Works, using the HC908QY4
http://mysite.verizon.net/bobsmith5/NITRON_Express.html
- The M68DEMO908QT4 Nitron Demonstration Board by Motorola

1.4 Xplor8 Source Code

Xplor8 is written in Object Pascal as implemented by Borland, in my case in Delphi versions 4 and 7. Two additional components are used in Xplor8 which are not natively present in my Standard copy of Delphi 4: The chief one is the comms port component and for this I've used the one developed by Dejan Crnila which is available as freeware on the SourceForge site. The other is an extended combo box available from the Tory's Delphi site. The Xplor8 source code is available from my web site if you wish to tinker.

1.5 Philosophy

1.5.1 Keyboard Input

Xplor8, although using the usual graphical interface, is not designed to be primarily mouse-driven. This is because my own experience of assembly language programming and debugging is that it calls for continuous use of the keyboard.

Default Hexadecimal Number Entry

I have also made the decision that, with very few exceptions, it is much easier to work all the time in hexadecimal. For this reason all commands default to expecting their data in hexadecimal (there is one exception where specifying a number of repeats - see the [Unassemble](#) command on page 62 for an example). Values may, however, also be input in decimal and in binary by the use of prefixes.

1.5.3 Screen Appearance

The actual GUI appearance of Xplor8 is loosely based on the on-screen appearance of Motorola's PCbug11. The extra resolution available with a modern graphical user interface has meant that certain features such as the breakpoint display may be permanently on view. The main form may be resized by dragging, and restored to one of three 'standard' sizes by clicking the menu item under View. Whatever size and layout the form has when it is closed will be remembered in the Windows Registry until next time. The Registry also stores the position and size of subsidiary windows opened as part of the program.

1.5.4 Command Syntax

The command syntax derives directly from my work on JBug11, and in general I've tried to make a combination of the best features of the debuggers that I'm familiar with. In the end it comes down to personal pride and the 'not invented here' syndrome. If you download the source code, you can modify the program to accept any command mnemonics you like.

1.6 Development History

Development of Xplor8 began around September 2002, with the first public release of version 0.0 in January 2003. Development then halted for many years, with the current version only being worked on in 2008.

1.7 Getting Started

For those simply wishing to get started as soon as possible, I've put the basic installation and operating instructions in [Appendix G](#).

1.8 Note on the Author

Before you, the reader, complain that the program doesn't appear to behave exactly as you would like, or lacks some essential feature, please bear in mind that I'm only a hobby programmer with an interest in the HC908 chips. I'm not working in a commercial environment. The upside of this is that the program is freely available, although I would of course much appreciate being told of any proposals or criticisms you may have. Let me know at john.beatty@virgin.net

1.9 Origin of the program logo

You may wonder why the program has a bundle of dynamite sticks as its logo. Of course, after you have used it for a bit, you may not wonder - but the simple explanation is this: when Motorola (as was before spinning Freescale off) introduced the QT1/QY4 series of HC908 processors they named them 'Nitron'. It was about then, in October 2002, that I began development of an interface with the HC908 monitor ROM program, and I called it NitroMon. This word sounded to me as though it had explosive connections, hence the logo. Motorola/Freescale subsequently dropped the 'Nitron' tag and I had to come up with another name for the interface program, and so 'Xplor8'.

1.10 An Appeal

Xplor8 has a large number of features, and may take a little time to set up. New users are advised to follow the Getting Started section in [Appendix G](#), and to follow the Quick Tour so as to get a feel for the program.

2 MISCELLANEOUS

2.1 Installing

Double click on the self-installing exe file, named something like 'Install-Xplor8-xyz.exe' which you have downloaded from the web site. This is a console application, so will start in a DOS box. You will be given the chance to change the default installation folder (C:\Program files\Xplor8) if you wish to install it somewhere else.

The installation utility does not also place an icon on the desktop; if want one, you can right click on Xplor8.exe and use the context menu to Send To>Desktop (create shortcut).

2.2 Uninstalling

To uninstall Xplor8, use the Windows Add/Remove Programs utility (in the Control Panel). This will remove the sub-directories of ..\Xplor8\ where they have no files in them other than those originally installed. If you have your own files in, for example, the \Project\ sub-folder, then these will need to be removed manually.

Uninstalling will not remove the Windows Registry entries, which are managed by Xplor8 itself, so if you need a completely clean machine, and are feeling brave, back up the Windows Registry and then delete the subkeys:

- HKEY_CURRENT_USER\Software\BeattSoft\Xplor8, and, possibly
- HKEY_USERS\Software\BeattSoft\Xplor8

2.3 Registry Entries

When Xplor8 is run for the first time on a new machine it will make an entry in the Windows Registry under HKEY_CURRENT_USER as follows:

HKEY_CURRENT_USER\Software\BeattSoft\Xplor8\1.x

This key will contain sub-keys that store information on the on-screen layout of the program, and on some aspects of the previous program launch, such as the name of the project file which was used last.

2.4 Project Files

Xplor8 stores the settings peculiar to each project in Project Files. These are plain-text files somewhat similar to old-fashioned Windows 'INI' files. They have the extension '.xp8', and this extension may be associated with Xplor8 so that double-clicking a project file will open it in Xplor8.

2.5 Supporting Files

To be fully operational, Xplor8 needs access to various supporting files of information. These are:

1. A configuration file for each version of the HC908 MCU range. This file records such data about the MCU as its memory map. Example name: 908QT1.cfg
2. Utility file(s) in standard Motorola S19 format to allow verification, programming and erasing of FLASH memory. Example name: QTY_UT1.S19
4. A file of information on the CPU opcodes. Default name: HC08_Opcodes.csv

This file is used by the [Unassemble](#), [Go \(Run\)](#) and [Trace](#) commands to establish and process break points. It is a text file in comma separated value format prepared directly from Freescale's published information on the binary codes for each mnemonic of their instruction set. It may be examined with any text editor (and changed at your own risk!).

5. A file of information on the MCU I/O registers. Example name: Regs_908QT1.csv

This file is used by the [Register Display and Change](#) command, as well as internally by Xplor8. These files have been prepared from the published Freescale documentation and may be examined in any text editor.

A full listing of the files in the distribution is in [Appendix H](#).

2.6 Remote Reset

Some target development boards allow remote control of the power supply to the MCU, so that a power-on reset (an essential prerequisite to entering Monitor Mode) may be initiated by Xplor8. This can save some time and frustration in program development. To make this possible, it is necessary to run a wire from one of the COM port control pins, either DTR (Data Terminal Ready) or RTS (Ready To Send) via some simple interface logic to control the MCU power supply; details are shown in [Appendix B - Hardware](#). When Xplor8 requires to reset the MCU, it toggles the polarity on the appropriate pin. See [Settings>COM Port](#) for information on setting up the toggles. A remote reset capability is a luxury - Xplor8 works perfectly well without it.

2.7 Monitor/User Mode Switching

With suitable circuitry on the target board, Xplor8 is able to make use of the spare COM port output line (either DTR or RTS, whichever is not in use by remote reset) as a general-purpose logic output to switch between monitor and user modes. If the circuitry is available (see [Appendix B - Hardware](#)), then this option may be set up in [Settings>COM Port](#), and activated using the right-hand speedbutton, or by the [Mode Switch](#) command ([page 52](#)). For an example of the use of this facility, see [Calibrating the Internal Oscillator](#) on .

2.8 Local and MCU-controlled Memory

The whole 64KB memory space of the MCU has its counterpart in a 64KB array of bytes within Xplor8. I refer to the latter array as the local version of MCU memory. It may or may not be an exact copy of the MCU memory, but is used internally whenever communication of memory contents with the MCU is required. For example, a command to list MCU memory will initiate a request for the

talker to transmit the required bytes of MCU memory to Xplor8 where they will be stored in the corresponding address locations in the local memory array. From there they will be displayed in the output window.

It should be noted that it is certainly not the case that the local memory is always a faithful copy of the MCU controlled memory. In fact, at start-up, the Xplor8 local memory is initially filled with zeros. Only as a result of commands such as L (List) will the local memory become a copy of the MCU memory. Also, commands which load the MCU with data, such as the command to load an S19 file, do so via the local memory, so making the local and MCU memories identical over the range of addresses covered by the loaded file.

Many commands have a version that affects only the local memory, for example 'LDL' - see [Load Memory](#) on page 51.

2.9 USB-to-Serial Adapters

Many USB-to-serial adaptors will not work satisfactorily with Xplor8 (or with other PC host programs). This is because they do not recognize the 'break' character emitted by the MCU when it enters Monitor Mode. As recognition of this break is critical to the operation of the host PC, such adaptors are useless.

However, adaptors based on the FTDI chipset appear to work just fine. The author uses a

'US232B/LC USB to RS232 Laptop Companion' from EasySync in the UK:

<http://www.easysync.co.uk/products.html>

or Saelig in North America:

<http://www.saelig.com>

FTDI's website is at: <http://www.ftdichip.com/index.html>

3 MON08 MONITOR FIRMWARE

3.1 Introduction

The HC908 series of micro controllers come with a small monitor program (MON08) already programmed into ROM. This program has a software bit-bang emulation of the standard RS232 serial protocol and six low-level commands which together allow the user to read and write RAM memory, read the stack pointer (SP) and run a program. It is the function of Xplor8 to interface with this monitor program and use its low level commands to perform realistic tasks such as programming FLASH memory.

To be able to enter this monitor program after resetting the MCU requires that the reset vector points to its start. As the normal reset vector (at \$FFFE/F) may have been programmed by the user to point to the start of the user's own program, the HC908 MCUs have a special mechanism to force the CPU to use different addresses for the reset and SWI vectors. This mechanism involves having certain voltages on various pins at the moment of reset - see any HC908 data sheet for details.

The monitor uses only a single pin on the MCU to communicate with the outside world. This means that full duplex communication is not possible, and also that every byte sent to the MCU from the host is inevitably also returned to the host as an unwanted echo. Xplor8 automatically disregards these unwanted echos.

The monitor program on the MCU provides six basic communication functions:

1. Read MCU memory
2. Write MCU memory
3. Indexed read of MCU memory
4. Indexed write of MCU memory
5. Read the Stack Pointer (SP)
6. Run a user program

All Xplor8's features are provided by using one or more of these six functions. When the CPU enters the monitor program it executes a software interrupt (SWI) instruction. This automatically stacks the program counter, A, X and CCR registers, and the monitor is so written that it also stacks the H register. This allows us to read and modify the CPU inherent registers before giving the 'run' command which unstacks H and then executes a Return from Interrupt (RTI).

Once a user program is running on the MCU there is no way for Xplor8 to seize back control, unless:

- The user program arrives at a pre-set breakpoint
- The user program arrives at an SWI instruction
- The MCU is reset

3.2 Utility Files

The on-chip monitor program has only basic commands which have to be used as the building blocks for more complex functions. In particular, the monitor 'write' commands can only write to RAM, so programming of Flash memory requires that another short program is available to do this. Such a program cannot itself reside in Flash, and therefore most HC908 chips have a Flash programming routine built into an auxiliary ROM area of memory. Xplor8 makes use of these ROM-resident routines where appropriate, and to do so it loads a small utility program into the RAM area whenever necessary. These utility programs are stored in S19 format files, and the source code is provided if you wish to understand better how they work.

3.3 Entering Monitor Mode

To enter monitor mode, the correct voltages must be applied to various MCU pins during reset. See the HC908 data sheet for the particular MCU that you are using. If these voltages are recognized, and the reset is effective (V_{dd} must drop below 100 mV before being restored), then the MCU will emit a 'break' character. When Xplor8 receives this break, it sends the eight security unlocking bytes as explained in the following section. Finally, Xplor8 reads a RAM memory location to determine whether or not unlocking has been successful.

3.4 Security Bytes

If it was simply a matter of applying the correct voltages to the pins of the MCU to enter Monitor Mode, then any user could read the contents of FLASH of any MCU. To make this much more difficult, it is arranged that eight bytes must be sent correctly to the MCU immediately following a power-on reset when it is desired to enter Monitor Mode. These eight bytes are compared with Flash locations (actually in the vector area) that have been programmed by the user. If the bytes do not match the MCU will still enter Monitor Mode, but all Flash memory locations read as garbage. The only command available in these circumstances is one to mass erase all of Flash memory.

The important consideration is that all access to FLASH memory is lost if the user doesn't know, or forgets, the values in the 8 security byte locations. Xplor8 therefore goes to some length to make it unlikely that these bytes will end up in an unknown state.

Security Byte Log

Xplor8 maintains a default log of all changes to the security bytes. Every time an operation is performed which might change these bytes, a record is added to the log which shows:

- Current project file name
- Date & time of modification
- The security bytes, and
- A note of the reason for the addition of the record.

The log is maintained in the same folder as the Xplor8 executable, and as a read-only file to help avoid accidental deletion. It is also backed up to a '.bak' file every time it is written. The file format is 'comma-separated value' (.csv) so it may be opened in a spreadsheet program such as Excel - but there should be no need for this as Xplor8 includes a viewer - see Security Bytes Log Viewer ([page 29](#)).

This viewer may be launched either from the View Menu menu or from Settings>Security Bytes ([page 38](#)).

The default log is limited to 100 records, and when the log reaches this size, the earliest records are progressively deleted as new ones are added. A warning is issued when the log is nearly full of new records, and the user is prompted to save a copy of the default log under a different name.

Entering Security Bytes

A new security byte string should be entered on the Settings>Security Bytes tab ([page 38](#)), or they may be entered as part of the Unlock command.

3.5 Baud Rates

All the HC908 chips that I have come across use a communication baud rate of 9600, when used with their standard crystals/oscillators. If you are using a non-standard crystal or oscillator, the baud rate may be pro-rated.

4 MEMORY MANAGEMENT

4.1 Writing to RAM

The monitor ROM routines on the MCU are able to write directly to RAM, including the I/O registers.

4.2 Writing to FLASH

Writing to (i.e. programming) FLASH memory is carried out transparently by Xplor8, subject to certain general conditions, as follows:

- Flash memory has been successfully unlocked
- The range to be programmed is not protected by a value in the FLBPR register - see the HC908 data sheet.
- The appropriate utility files are available

If the above conditions are met, Xplor8 loads the necessary utility file to the RAM of the MCU and sends the data. Where the new data overlaps already-programmed locations, Xplor8 automatically manages the reading-in and rewriting of memory which is not receiving new data, but which otherwise has to be erased to allow new data to be written.

4.3 Managing the FLBPR Register

On most chips in the HC908 series, a FLBPR (Flash Block Protect) register provides some protection against accidental erasure or change of Flash contents. On some flavors of the HC908 this register is implemented as a Flash byte, while on others it is a RAM location that reverts to \$00 (all locations protected) on reset. It is the user's responsibility to write the FLBPR register to an appropriate value before attempting a command to modify Flash memory. As an example, the following macro will unprotect Flash, on a chip where FLBPR is a RAM location, erase all the Flash locations, re-protect memory and verify that it is indeed fully erased:

```
DEFM      EraseAll
BEGIN
    R FLBPR=FF
    ME FFFF
    R FLBPR=00
    VE
END
```

4.4 Memory Map Display

This window displays the memory map for the currently selected MCU. It reflects the data supplied in [Settings>Memory](#).

The map may be sorted with the 'sort' button either in ascending or descending order. This window is launched from the View menu, or by the keyboard shortcut Ctrl+M.

5 DEBUGGING

5.1 Breakpoints

Tracing, and running programs to breakpoints, requires that breakpoints can be established at the right places in the MCU program code. On the HC908 chips only a single breakpoint can be set at one time, by writing a value to the breakpoint register. Programs may be run or traced in both RAM and FLASH memory. When tracing at a branch instruction, Xplor8 calculates where program execution will continue by its own analysis of the condition code register, and sets the next breakpoint accordingly.

A breakpoint cannot be set at:

- illegal opcodes
- indexed jump instructions
- the SWI instruction
- BIH or BIL instructions
- any instruction which jumps or branches to itself

A breakpoint may be set or deleted as part of the Go (Run) command. The Trace command automatically sets the breakpoint. When a breakpoint address has been written, the Breakpoint Register display on the main form is updated.

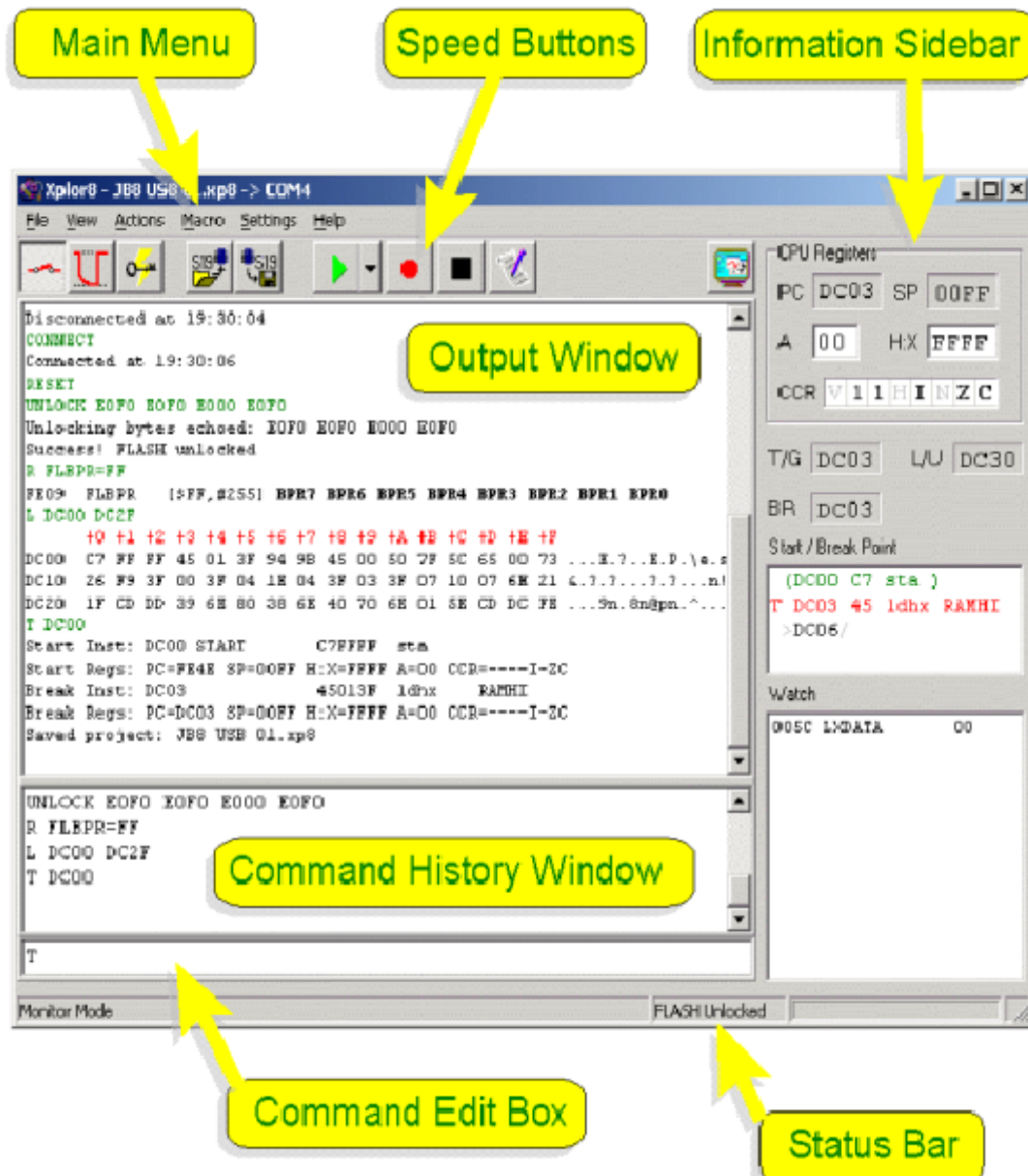
The breakpoint module in the HC908xx MCU's has some quirks, probably due to the instruction pre-fetch architecture. You may find that setting breakpoints at branch destinations or at RTS instructions results in a running program stopping at unexpected places. This odd behavior does not appear to affect tracing. The \Samples\ subdirectory contains a program which demonstrates this.

5.2 SWI in user code

If a Software Interrupt (SWI) instruction is encountered by a running program, while in monitor mode, control will return to the monitor and this is useful for testing code segments. When control does return to the monitor in these circumstances, the output window will display the message: 'unexpected break received'.

6 PROGRAM FEATURES

6.1 Screen Layout



6.2 Main Menu

The main menu provides access the various facilities in Xplor8. For more details see the individual menu descriptions later in this chapter.

6.3 Speedbuttons

Nine speedbuttons are provided along the top of the main form to carry out common tasks. From left to right these are:

Connect	Click to carry out the Connect / Disconnect command (page 43), i.e. open or close the serial (COM) port.
Reset	Click to carry out the Reset command (page 56). This button will be grayed-out if remote resetting is not available.
Unlock	Click to send the eight security bytes by means of the Unlock command.
Load S19 file to MCU memory	Click to carry out the Load Memory command (page 51). This button is grayed-out unless FLASH is unlocked, and the status line reads 'Monitor Mode'.
Save MCU memory in S19 file	Click to carry out the Save Memory command (page 57). Grayed-out as above.
Play Macro	Click to play the last macro selected from the drop-down list. If a macro is already playing, click this button again to stop it (or use the Stop button).
Record Macro	Click to begin recording a macro. If a macro is being recorded, click this button again to stop recording (or use the Stop button).
Stop	Click to stop a macro playing or recording.
Edit Macro	Click to open the Macro Editor . (See page 72)
Switch between Monitor and User modes	Click to operate the remote switch, see General-Purpose Switching on page 11. This button is only available if remote mode switching is enabled - and this requires the necessary hardware on the target board.

6.4 Output Window

Displays commands and their results. Also miscellaneous information, such as diagnostic information following communication errors.

The division between the output and command history windows is a moveable splitter bar, allowing one window to be enlarged or reduced at the expense of the other. Word wrap is enabled in this window.

A right-click context menu is available within the Output Window. This provides:

- **Select All**
- **Copy**
- **Delete**
- **Clear All**

6.5 Command History Window

Displays a list of all the commands issued during the use of the program.

A command shown here may be re-used by:

- clicking on it, whereupon it will replace whatever currently appears in the [Command Edit Box](#), or by
- double clicking, when it will be executed immediately, or by
- using the up and down arrow keys while the Command Edit Box has focus will recall a previous command.

Note that if a previously issued command is selected, it will be transferred to the Command Edit Box without any comment that may have been added (see [Commands](#) on page 40 for an explanation of the addition of comments).

A right-click context menu is available within the Command History Window. This provides:

- **Help on Error Message** If a previous command has resulted in an error message in the Command History Window, of the type that begins with an arrow: <--, and this line has been highlighted by left-clicking on it, then this menu item will open on-line help at the relevant topic. This menu item is not available if more than one line has been highlighted or the line does not have an error.
- **Select All**
- **Copy**

6.6 Command Edit Box

This is a standard Windows edit box for the typing-in of commands. It is designed to be the main point of user interaction with Xplor8. Pressing the Enter or Return key while this box has focus initiates the following events:

1. The command is parsed and checked for syntactical correctness. If incorrect, an explanatory message is appended to the command.

2. The command is copied to the command history list and to the output window, where it appears in green type.
3. If the command is correctly formatted, it is executed and the results, if any, will appear in the Output Window and in the [Information Sidebar](#) displays, as appropriate.

Pressing the escape key while this box has focus will clear its contents, and stop any macro playing or recording.

6.7 Status Line

This is divided into three panels as follows (from the left):

1. General program state

This panel will display one of the following messages depending on the current program state:

Invalid project data (check "Settings")
Disconnected
Connected
Monitor Mode
Running
Tracing
User Mode

2. Locked/Unlocked

This panel shows the result of sending the security bytes - whether flash memory was successfully unlocked or not..

3. Progress Bar

This is a standard windows progress bar which shows the progress of operations that involve the transfer of bytes to or from the MCU.

6.8 Information Sidebar

The right-hand side of the Xplor8 main form is given over to a display of various items of information useful when tracing and running programs on the MCU. Resize this area using the vertical splitter bar. From top to bottom, information is displayed as follows:

6.8.1 CPU Registers Display

Whenever Xplor8 returns to monitor mode, that is to say directly interfacing with the on-chip MON08 firmware, the display of the MCU inherent register values is updated. So, for example, when stopping at a breakpoint, these boxes will show the current program values in these registers. The A and H:X boxes are editable so that their values may be changed - useful if you are tracing and want to try alternatives. The flags in the CCR box may be toggled by double-clicking them. The PC and SP (stack pointer) are read-only. The PC can only be changed as part of the Go (Run) or Trace commands.

6.8.2 T/G, L/U and BR addresses

- **T/G** The T/G address shows where Tracing and Running will next begin in the absence of a specific address
- **L/U** The L/U address shows where Listing or Unassembling will begin in the absence of a specific address
- **BR** The BR address shows the latest value written to the BRKH/L registers on the MCU.

Suppose a file to be loaded at \$8000 and the following Trace command given:

```
T 8000
```

then tracing will commence at 8000 (hex), and when tracing stops at the next breakpoint, the value of the T/G address will be updated to show the break point address. Simply pressing the <Enter> key will then trace the next instruction. The L/U address also tracks the breakpoint, so if the program were paused at a breakpoint, say \$8003, then issuing the command:

```
U3
```

would unassemble the next three instructions, altering L/U to the address immediately after the operands of the third instruction, but leaving T/G unchanged, so that typing

```
T
```

would resume tracing where it was previously left off.

6.8.3 Start / Break Point

Displays breakpoint information. When running or tracing, the top line in this window shows the instruction at the starting address in brackets and in green, followed by the currently set break point (if any). If the MCU is halted at the break point, then that line is displayed in red type, followed by a line showing where the program would go next if another trace command is given. If halted at a branch instruction, two addresses will be displayed, the sequential one first and then the branch address. Whichever is the current destination will be highlighted in bold.

6.8.4 Watch Window

Displays the value of selected memory locations during the running and tracing of programs. The items in the watch window are automatically updated when a running or tracing program returns to monitor mode.

Items in this display may be re-ordered by dragging and dropping.

A right-click context menu is available in this display, with the following items:

Refresh	Click to read the current value(s) at the address(es) and update the display in the watch window.
----------------	---

Add...	Click to open the Add to Watch dialog for adding an address to watch (see below).
Delete	Deletes an item selected in the Watch display.
Clear All	Deletes all the items in the display.
Undelete	Restores deleted items

Adjust the relative sizes of the Start/Break Point display and the Watch window with the moveable splitter bar. The addresses and labels in the Watch display are remembered in the project file between sessions.

6.9 Add to Watch

Dialog for adding items to the watch window. The dialog allows you to add addresses in different categories, as follows:

User Defined

Type the address in MCU memory that you wish to watch in the 'Address' edit box (hexadecimal characters only), and add an optional description. Click 'Add' to add this item to the watch window, or 'Add & Close' to add the item and simultaneously close the 'Add to Watch' dialog.

Stack Relative

Addresses relative to the stack pointer may be watched. Type in an offset or use the drop-down list. The default offset number base is decimal, hex numbers may be entered with a \$ prefix. Zero is an acceptable offset, but not negative amounts.

Index Relative

As for 'Stack Relative' above, but in this case the offset is with respect to the H:X register pair.

Go to Symbols/Registers

Click to close the dialog and open the [Symbol Table/Register Display](#). From this display, symbolic labels may be added to the watch window.

6.10 File Menu

Menu used to manage [project files](#).

New Project...	Begins a fresh project. If the current project has changed, the user is prompted to save it first, then the Settings Dialog opens with default values
Open Project...	Opens, and makes current, an existing project configuration. Keyboard shortcut: Ctrl+O

Save Project...	Saves the current configuration to a project file. If the file exists, it will be silently updated, if not, a file save dialog will appear. Keyboard shortcut: Ctrl+S
Save Project As...	Opens a file save dialog for the user to save the current configuration to a file with a new name. Keyboard shortcut: F12
Reopen...	The sub-menu displays a list of up to ten of the most recently used project files. The top entry in the list is grayed-out if it is the one currently open.
Import...	Open a dialog to Import configurations from a Xplor8 version 4xx. This item is grayed out (unavailable) if no version 4xx configuration information is found in the Windows Registry.
Exit	Closes Xplor8. If the current project configuration information has changed, the user is prompted to save it before closing. Similarly, if the Macro Editor has been in use, and the results not saved, the user will be reminded to save them.
Terminate	Terminates the program in circumstances where exit does not appear to work.

6.11 View Menu

Base Converter...	Opens the number Base Converter (page 28). Shortcut: Ctrl+K
Macro Editor...	Opens the Macro Editor (page 72). Shortcut: Ctrl+E
Symbol Table...	Opens the Symbol Table/Register Display (page 28) with the currently-loaded symbol table on display. Shortcut: Ctrl+L
I/O Registers...	Opens the Symbol Table/Register Display (page 28) with the I/O register list on display. Shortcut: Ctrl+R
SB Log...	Opens the security byte log viewer (page 29). Shortcut is Ctrl+H.
Terminal...	Brings up the Terminal Window (page 74). Shortcut is Ctrl+T.
Memory Map...	Opens the Memory Map Display (page 16). Shortcut: Ctrl+M
Base Layout	Use the sub-menu to select a starting point layout for various screen resolutions. The actual layout is remembered in the Windows registry at the end of each session, so the program should have the same on-screen appearance when you next start it up.
Font Sizes	Chose a font size for the Output and Command History windows.

6.12 Actions Menu

Menu to simplify the issuing of commands that involve the loading and saving of files to and from the MCU. Such commands all need a file name typed on the command line which is tiresome to do accurately. Using these menu items brings up a 'file open' or 'file save' dialog as appropriate, in which the user may select a file name in the usual way. Closing the dialog then fills in the command line and executes the command. Actions involving transfers to and from MCU-controlled memory will be grayed out (unavailable) unless FLASH is unlocked and the status line reads 'Monitor Mode'.

Connect	Executes the Connect / Disconnect command (page 43) to open the serial port (COM port).
Reset	Executes the Reset command (page 56) to send a reset signal to the MCU. Note that this is only available if your hardware supports it, and the COM port is connected.
Unlock	Executes the Unlock (page 65) command to send the security bytes to the MCU. Not available when disconnected. Shortcut: Ctrl+U
Load S19 MCU...	Opens a dialog for the user to select an S19 format file for loading to the MCU. The Load Memory (page 51) command is then executed via the command line.
Save MCU to S19...	Saves a block of MCU memory to a file in S19 format. An address input dialog will appear for the user to define the starting and ending addresses of the block to be saved, followed by a file-save dialog. The Save Memory command (page 57) is then executed via the command line.
Verify S19...	Opens a dialog for the user to select an S19 format file for verifying MCU memory. The Verify command (page 66) is then executed via the command line.
Load Binary to MCU...	Opens a dialog for the user to select a binary image file (usual extension: .obj or .bin). A second dialog box appears to request a loading address. If a valid file is selected, the file is opened and loaded to MCU controlled memory via the Command edit box, as though the Load Memory command (page 51) had been issued.
Save MCU to Binary...	Saves a block of MCU-controlled memory to a binary image file. An address input dialog will appear for the user to define the starting and ending addresses of the block to be saved, followed by a file-save dialog. The Save Memory command (page 57) is then executed via the command line.

Local

Sub-menu allows the user to load, save and verify S19 files against [local memory](#). These commands do not require an MCU to be connected.

Load S19 to Local...	As though the 'LDL' command had been issued - see the Load Memory command
Save Local to S19...	As though the 'SVL' command had been issued - see the Save Memory command
Verify S19 v. Local...	As though the 'VL' command had been issued - see the Verify command
Load Binary to Local...	As though the 'LDL' command had been issued - see the Load Memory command
Save Local to Binary...	As though the 'SVL' command had been issued - see the Save Memory command

6.13 Macro Menu

Boot Script...	Open the Settings>Macros tab for editing the Boot Script (see page 69).
New...	Begin a new macro library script. If the currently open one has changed, the user will be prompted to save it first. The contents of the macro editor will be cleared and the editor window shown if not currently visible.
Open ...	Open a macro library file into the Macro Editor . The default file extension is '.mcr'. Shortcut is Ctrl+O.
Save	Save the contents of the macro editor to a macro library file. Shortcut is Ctrl+S.
Save As...	Save the contents of the macro editor to a new file, default extension '.mcr'.
Record	Start recording, adding new commands to the end of the current macro library.
Play	Choose a macro to play from the sub-menu.
Stop	Stop playing or recording.
Edit	Open the Macro Editor

6.14 Settings Menu

Open a tab in the [Settings Dialog](#). Once the Settings dialog is open, any other tab may be selected.

6.15 Help Menu

Index	Opens Help Topics at the Index tab. Shortcut is F1.
--------------	---

Contents	Opens Help Topics at the Contents tab. Shortcut is Shift+F1.
Getting Started	Opens Help at 'Getting Started' - see Appendix G .
Error Report...	Opens the Error Report dialog for compiling a text file report on a problem encountered while using Xplor8, see page 79 . Useful if email support is required.
About	Brings up the usual 'About' box.

6.16 Keyboard Shortcuts

Keyboard Shortcuts - Main Window Active

Ctrl+B	Reset the MCU. Useful if you have fitted the remote resetting hardware.
Ctrl+E	Open the Macro Editor
Ctrl+H	Opens the Security Bytes viewer
Ctrl+K	Opens the number Base Converter
Ctrl+L	Open the Symbol Table/Register Display with the currently-loaded symbol table on display.
Ctrl+M	Opens the Memory Map Display .
Ctrl+O	Open a project file
Ctrl+R	Opens the Symbol Table/Register Display with the MCU register list on display
Ctrl+S	Save the current project file
Ctrl+T	Open the Terminal Window .
F12	Save the current project file under a new name
Esc	Clear the command line (if it has focus). If a macro is playing, it will stop.
F1	Open on-line help at the 'Index' tab
Shift+F1	Open on-line help at the 'Contents' tab

Keyboard Shortcuts - Macro Editing Window Active

Ctrl+O	Open a macro library file into the macro editor
Ctrl+S	Save the current macro library file

Ctrl+C	Copy to Clipboard
Ctrl+V	Paste from Clipboard
Ctrl+X	Cut to Clipboard
Ctrl+Y	Delete the line containing the insertion point (caret).

Keyboard Shortcuts - Terminal Window Active

Ctrl+I	Send a binary image file
Ctrl+Q	Close the terminal window

6.17 Symbol Table/Register Display

Window to display the currently-loaded symbol table (see [Using a Symbol Table](#) on page 63), or the current MCU control registers.

Display this window by clicking the item 'Symbol Table' or 'MCU Registers' in the View menu, or use the keyboard shortcuts Ctrl+L or Ctrl+R.

- Select which kind of display you want with the buttons at the top of the form.
- Sort the list by address or name using the radio buttons at the foot of the window.
- Select one or more items in the list and click 'Add to Watch' to transfer them to the Watch window.

6.18 Base Converter

This window allows rapid conversion between hexadecimal, decimal and binary notations. Launch it from the view menu, or by using the keyboard shortcut Ctrl+K.

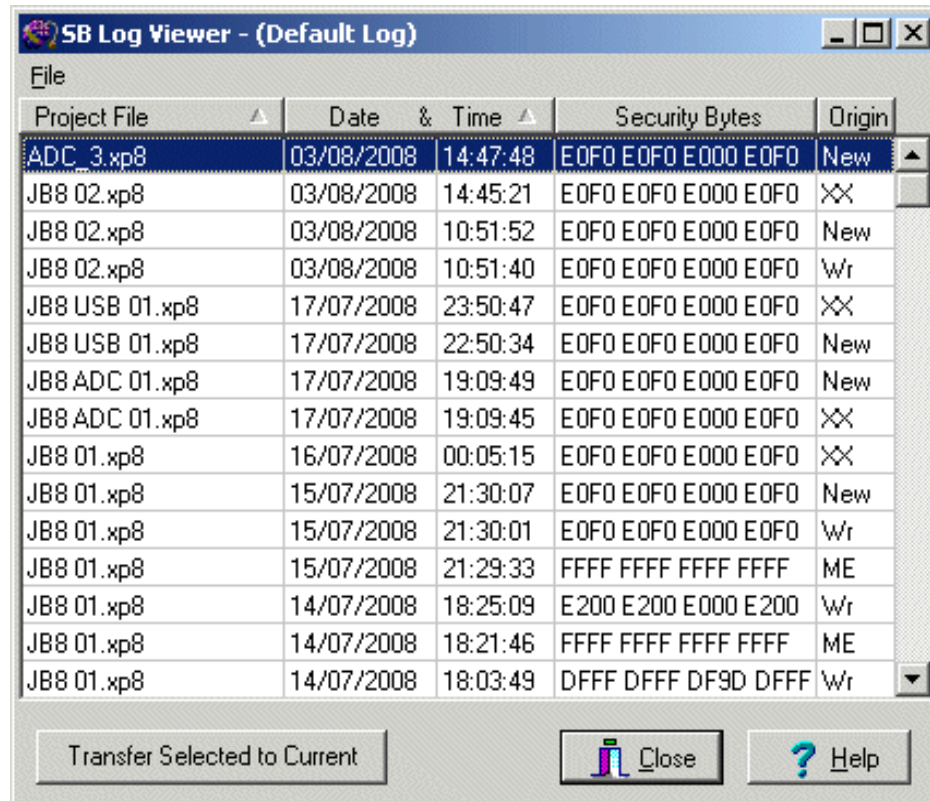
Entering a valid number in one of the boxes and pressing the enter key, or clicking the '=' button, will fill in the other two boxes accordingly. If the 'Auto completion' check box is ticked, the current hexadecimal number in the calculator will be appended to the text in the Command edit box, and focus will be transferred to that box. The converter form will not close, but will become inactive. If 'Auto completion' is unchecked, then clicking '=' will not affect the command line, and will leave the converter open and active.

The maximum value that the converter will handle is \$FFFF or 65535 decimal. Only valid hexadecimal characters may be entered in the 'Hexadecimal' edit box, similarly for the other two boxes.

Clicking the 'ASCII' button brings up a quick-reference table of ASCII characters (in the Arial font face). Double-clicking on a character in this table will transfer its hexadecimal value to the converter.

6.19 Security Bytes Log Viewer

This window displays a log of security bytes. It may be used to view the default log, or one of the copies saved by the user. For general details of how Xplor8 handles security bytes see Security Bytes (page 14).



Project File	Date	Time	Security Bytes	Origin
ADC 3.xp8	03/08/2008	14:47:48	E0F0 E0F0 E000 E0F0	New
JB8 02.xp8	03/08/2008	14:45:21	E0F0 E0F0 E000 E0F0	XX
JB8 02.xp8	03/08/2008	10:51:52	E0F0 E0F0 E000 E0F0	New
JB8 02.xp8	03/08/2008	10:51:40	E0F0 E0F0 E000 E0F0	Wr
JB8 USB 01.xp8	17/07/2008	23:50:47	E0F0 E0F0 E000 E0F0	XX
JB8 USB 01.xp8	17/07/2008	22:50:34	E0F0 E0F0 E000 E0F0	New
JB8 ADC 01.xp8	17/07/2008	19:09:49	E0F0 E0F0 E000 E0F0	New
JB8 ADC 01.xp8	17/07/2008	19:09:45	E0F0 E0F0 E000 E0F0	XX
JB8 01.xp8	16/07/2008	00:05:15	E0F0 E0F0 E000 E0F0	XX
JB8 01.xp8	15/07/2008	21:30:07	E0F0 E0F0 E000 E0F0	New
JB8 01.xp8	15/07/2008	21:30:01	E0F0 E0F0 E000 E0F0	Wr
JB8 01.xp8	15/07/2008	21:29:33	FFFF FFFF FFFF FFFF	ME
JB8 01.xp8	14/07/2008	18:25:09	E200 E200 E000 E200	Wr
JB8 01.xp8	14/07/2008	18:21:46	FFFF FFFF FFFF FFFF	ME
JB8 01.xp8	14/07/2008	18:03:49	DFFF DFFF DF9D DFFF	Wr

On the “File Menu” are the following options:

Open Default Log Click to display the default security byte log. This is the prime location updated every time Xplor8 detects a change in the security bytes

Open Saved Log... Click to open a previously saved copy log.

Save Default As... Click to save a copy of the default log in a file under a new name.

Close Click to close the SB Log Viewer.

Display Window

The bulk of the window is taken up by a display of the log organized into five columns. Click on the 'Project File', 'Date & Time' or 'Origin' columns to sort them up or down. When a new log file is first opened, the records are always sorted in date/time order so that the most recent record is at the top. The 'Origin' column provides information about the source of the record, as follows:

Wr The record is a result of a write operation covering the memory locations occupied by the security bytes. Note that the bytes which appear in the record were read in automatically by

Xplor8 after the write operation and are not simply the bytes which the user sent to program memory.

New The user has successfully attempted an Unlock operation with a new entry in the 'Current Security Bytes' edit boxes in Settings>Security Bytes (page 38). 'New' is taken by Xplor8 to mean not previously noted in the default log file.

XX An attempted Unlock operation with these bytes failed.

Rd The user clicked the 'Read off MCU' button in Settings>Security Bytes.

PE The page containing the security bytes was erased.

ME A mass erase operation was performed.

Transfer Selected to Current

Click this button to make the bytes of the selected record the 'Current Security Bytes' (see Settings>Security Bytes, page 38). Double-clicking a record in the display has the same effect.

7 CONFIGURATION

7.1 Settings Dialog

The six tabs of the Settings Dialog contain various editable fields for the customization of Xplor8. All the configuration data on the tabs is stored in [project files](#) when the program closes, and the file name of the project file is itself stored in the Windows Registry, so that the settings are restored when Xplor8 is started again. At the bottom of the dialog are three buttons:

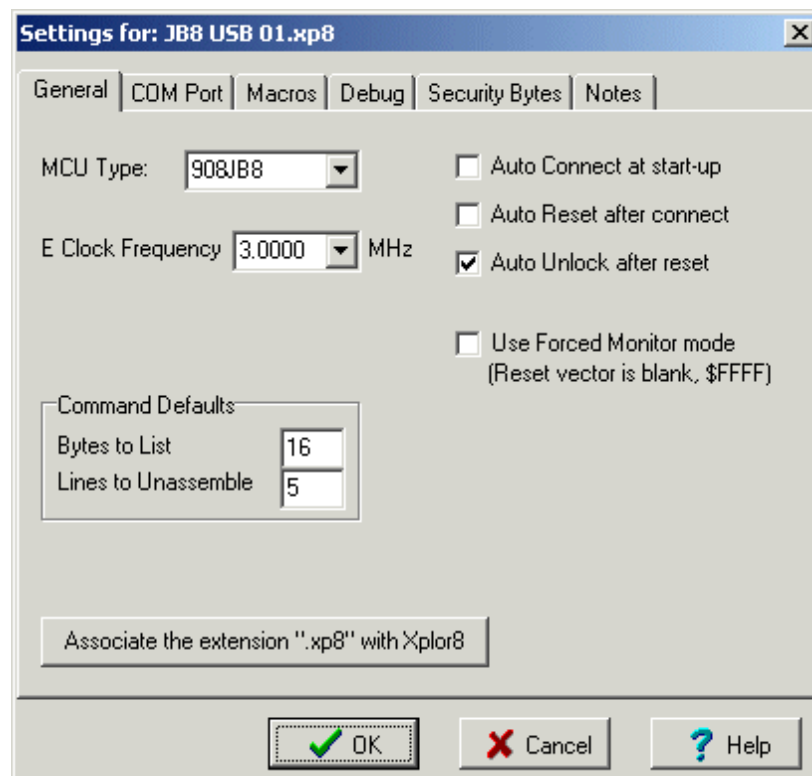
OK Button Click to close and save the data entered in the various tabs. The data is checked for consistency, and if incorrect or missing data is found, an error message will appear with diagnostic information.

Cancel Button Click to close Settings and discard any changes

Help Button Click to open the on-line help topic corresponding to the currently visible tab.

7.2 Settings>General

Tab for miscellaneous, general settings.



MCU Type

Select the MCU that you are using from the drop-down list. This action will fill in all the relevant edit boxes in 'Settings' with the default information for this MCU. This information is taken from the file MCUData.cfg in the 'MCU' subdirectory of the installation folder. If the particular MCU that you are using is not in the drop-down list, then it may be added to this file - see [Appendix D](#).

E Clock Frequency

Select the E clock frequency at which the MCU is working. See the HC908 data sheet for the chip that you are using. Those chips with an internal RC oscillator may be operating on either their internal oscillator or on an external crystal.

Command Defaults

Bytes to List Sets the default value for the number of bytes to list if the [List](#) command (page 50) is issued with only a starting address.

Lines to Unassemble Sets the number of instructions to unassemble if the [Unassemble](#) command (page 62) is issued with only a starting address.

Auto Connect at start-up

Tick this box so that Xplor8 automatically issues the Connect command when the program is launched - see the [Connect / Disconnect](#) command on page 43.

Auto Reset after connect

Tick this box so that Xplor8 automatically issues the Reset command after connecting.

Auto unlock after reset

Tick this box to have Xplor8 automatically issue the Unlock command after resetting.

Use Forced Monitor mode

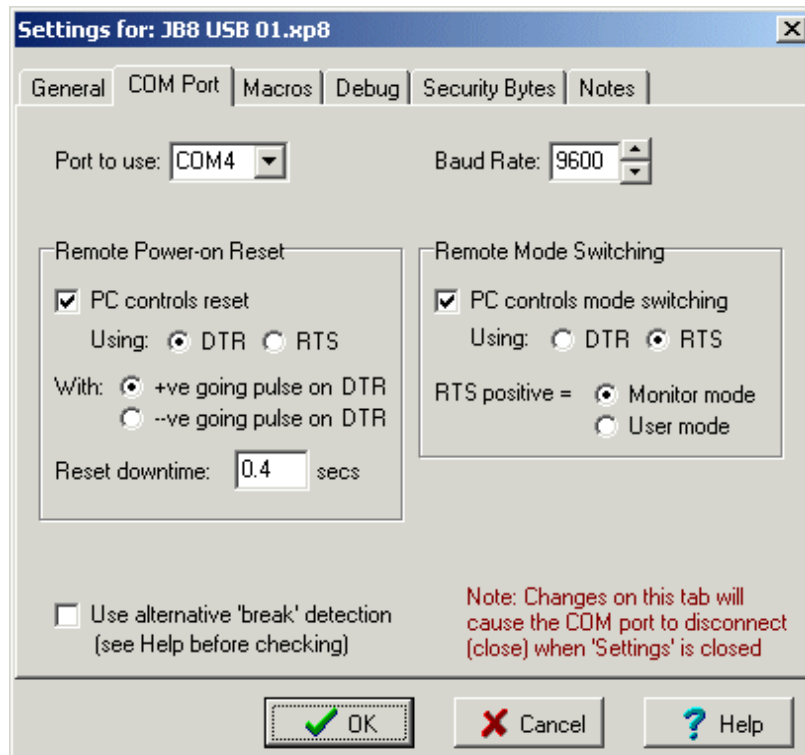
Tick this box if working with a fully-erased (or brand new) chip.

Associate the extension ".xp8" with Xplor8

Click this button to do as its caption says. When Xplor8 is first run, it automatically makes this association, but if there is subsequently a problem with file associations, the link between the .xp8 project file type and the Xplor8 executable may be restored with this button.

7.3 Settings>COM Port

Tab for configuring the RS232 port which will be used to communicate with the target board. Note the caveats in [USB-to-Serial Adapters](#) on page 12.



Port to use:

Select an available COM port from the drop-down list. This combo box displays all the COM ports returned by the Windows 'Enumerate COM Ports' function.

Baud Rate

This combo box selects the baud rate which Xplor8 uses to communicate with the Monitor ROM program on the MCU. Unless you are using a non-standard crystal it will usually be 9600 baud.

Note that the only baud rates obtainable in practice are those which are integer divisions of 115200 baud, for example 7680 baud is obtainable because $7680 = 115200 \div 15$. If a rate somewhere between the available rates is entered in one of the edit boxes, then the UART in the PC defaults to the first available rate **above** the rate entered. Beside the box is a spin control which will automatically select the next higher or lower integer division of 115200.

Remote Power-on Reset

The options on this panel control the signaling of a reset between the host PC and the target board. The RS232 communication ports on a PC have two available output pins (besides TxD): DTR (Data Terminal Ready) and RTS (Ready To Send). Either of these may be used to perform a reset remotely, provided the necessary hardware logic is in place (see [Appendix B - Hardware](#)).

PC controls reset

Tick this box if the host PC is able to control the reset cycle on the target board.

Using DTR/RTS

Select whichever signal is used for the remote re-setting.

With +ve/-ve going pulse

Select the pulse polarity. For the avoidance of doubt about the meaning of RS232 'asserted', etc. the descriptions +ve/-ve refer to the actual polarity of the pulse with respect to the common ground pin (pin 5 on the DB9 connector). Note that most RS232 level shifter chips on target boards invert this polarity, and also that RS232 line receivers usually bias their inputs so that a disconnected input appears to be at a negative voltage.

Reset downtime

Time, in seconds, for the reset pulse to remain in the 'low' state. The minimum value is 0.1 s (100 ms), and the maximum 999 s (16 min 39 sec - a long wait!). The important thing is that Vdd falls to within 100 mV of Vss to achieve a reliable reset. The default value is 0.1 seconds, but this may well be too short - it all depends on the target board circuitry.

Remote Monitor/User Mode Switch

With the necessary logic hardware on the target board, the PC can be used to control whether the MCU runs in Monitor mode, or executes its own user-written program; using whichever RS232 output signal is not already being used by the remote reset function. See [Monitor/User Mode Switching](#) on page 11 and [Appendix B - Hardware](#) for more details.

PC controls switching

Tick this box if the host PC is able to toggle the output.

Using:

Select whichever RS232 output pin is to control the switching. If DTR is selected then the labels in this box will change to DTR rather than RTS.

General-purpose switch is 'ON' when RTS is positive/RTS is negative

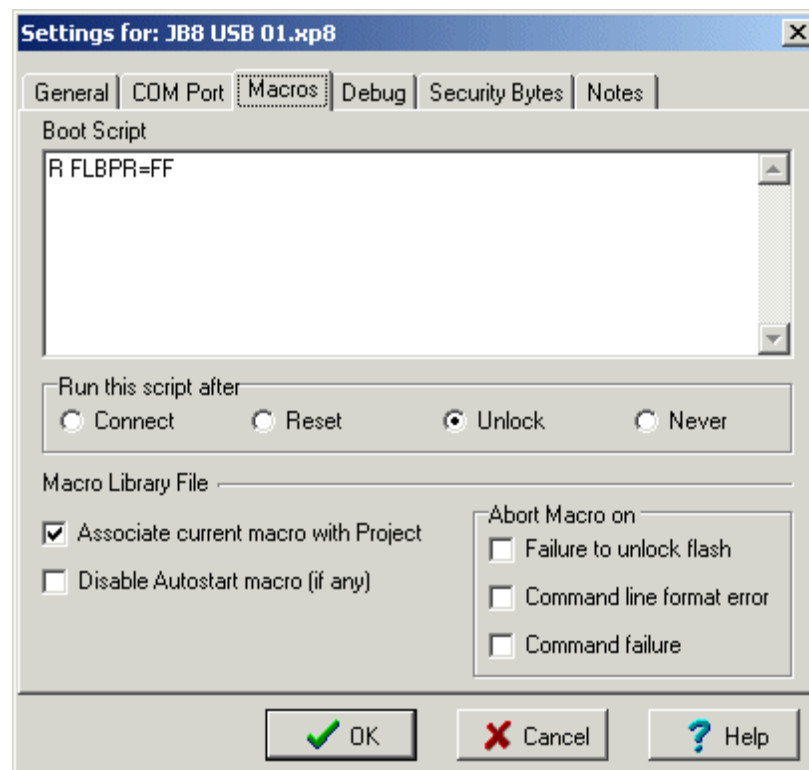
Choose the pin voltage level which corresponds to the 'ON' state.

General

Note that changes made on this tab will cause the COM port to close (disconnect) if the Settings dialog is closed by clicking the 'OK' button.

7.4 Settings>Macros

Tab for editing the boot script and for settings affecting macro library files.



Run this script on boot

Tick this check box and enter the desired commands in the edit box below. See [Boot Script](#) on page 69 for more details.

Associate current macro with Project

Check this box to cause the current macro library file to be reloaded when the current project is reloaded. The current macro library file is whatever is currently open in the [Macro Editor](#) (page 72) when the project file is saved.

Disable Autostart macro (if any)

Check this box to prevent any macro named 'AUTOSTART' from running - see [Autostart Macro](#) on page 70 for details.

Abort Macro on

A playing macro can be arranged to stop automatically under certain conditions. These are:

Command line format error

Tick this box to cause a playing macro to stop if a command within the macro is formatted incorrectly, causing an error message beginning with the <-- symbol to appear in the Command History window.

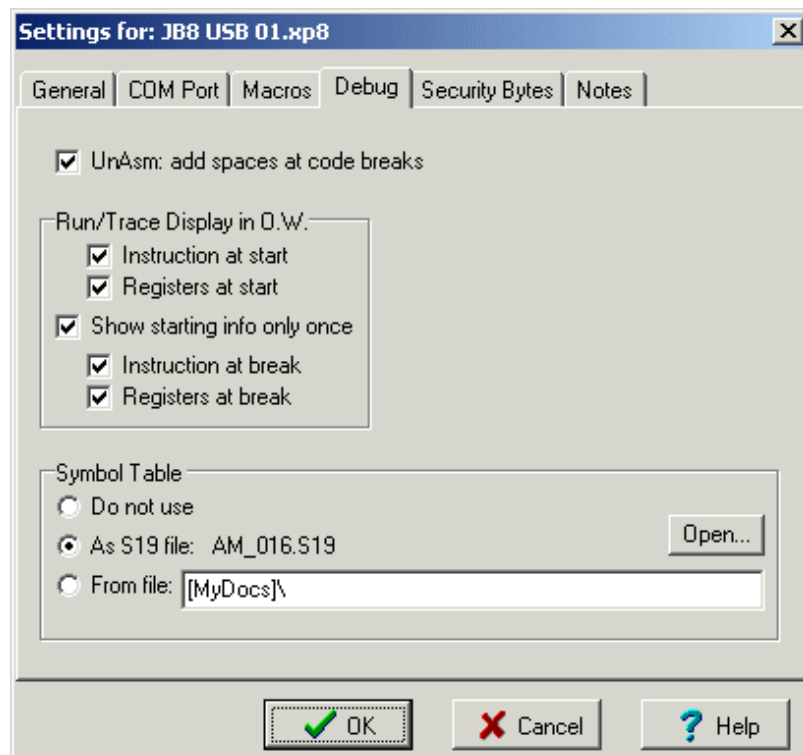
Command failure

Tick this box to cause a playing macro to stop if a command within the macro has an unsuccessful result; for example the [Verify Erase](#) command (page 67) finds some un-erased memory.

7.5 Settings>Debug

Tab for configuring the behavior of Xplor8 while running and tracing programs, including:

- How much information is displayed in the Output Window,
- How SWI instructions and illegal opcodes are treated, and
- The behavior when the [Step Over](#) command (page 61) is used.



UnAsm: add spaces at code breaks

Check this box to have Xplor8 add a blank line in unassembled listings after the following instructions:

- Unconditional jumps, including the BRA (Branch Always) instruction. No space is left if the jump address is that of the immediately following instruction, i.e. the jump was included simply to use up processor cycles.
- Return from Interrupt, RTI
- Return from Subroutine, RTS

Such spaces can improve the readability of disassembled code by highlighting the places where sequential code execution is not possible.

Run/Trace Display in O.W.

This group of checkboxes governs the extent of the information provided in the output window during running and tracing.

Instruction at Start : check this box to display a disassembly of the CPU instruction at the address at which running or tracing started.

Registers at Start: check this box to add a display of the CPU inherent registers at the start point.

Show starting info only once: with repetitive tracing there is no need to display the starting **and** break information, since the break information at one step will be the same as the starting information for the next step. Check this box to suppress the additional information.

Instruction at Break: check this box to display a disassembly of the CPU instruction at the breakpoint address.

Registers at Break: check this box to add a display of the CPU inherent registers at the breakpoint.

Symbol Table

When tracing or disassembling code, symbolic labels may optionally be displayed. This panel controls the display, and the location of the symbol table from which the information is taken. See [Using a Symbol Table](#) on page 63 for more details.

Do not use Symbolic labels will not be added to traced or disassembled code.

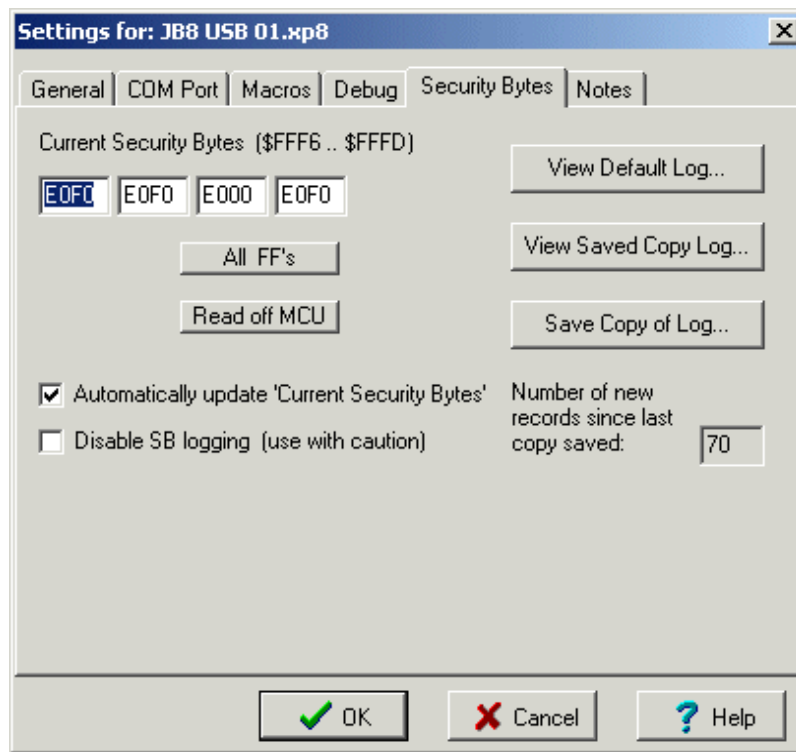
As last loaded S19 file (plus filename, if one is loaded)

Select this option if symbolic label information is to be taken from a symbol table generated alongside an S19 file. If selected, then every time an S19 record file is loaded to memory, Xplor8 will look to load a symbol table of the same name but '.sym' extension.

From file: Select this option if symbolic label information is to be taken from a fixed location. If so, choose a file in the edit box. File names may appear in this edit box with a token replacing part of the path - see [Path Tokens](#) on page 41.

7.6 Settings>Security Bytes

Tab for working with the security bytes needed to unlock FLASH memory. See Security Bytes (page 14) for a full explanation of how Xplor8 handles these.



Current Security Bytes (\$FFF6..\$FFFD)

Enter the bytes (as four 2-byte hexadecimal words) into the edit boxes. These bytes will be sent by default by the Unlock command. The bytes are grouped into words to correspond with the vectors which reside at the addresses \$FFF6 to \$FFFD.

All FF's

Click this button to fill all the Current Security Bytes boxes with FFFF, the values appropriate to a fully-erased device.

Read off MCU

Click this button to read the current security bytes off the MCU. This button is greyed-out (unavailable) if a successful FLASH unlock operation has not already been performed on the MCU.

View Default Log...

Click to open the SB Log Viewer to display the default log of security bytes maintained by Xplor8.

View Saved Copy Log...

Click for a file-open dialog to select a previously-saved copy of the security byte log. The selected file is then displayed in the SB Log Viewer.

Save Copy of Log...

Click to open a save-file dialog so that you may save a copy of the default security byte log under a different name.

Number of new records since last copy saved:

Read-only edit box displaying the number of read/write records saved in the default log since a copy was last made by the user. A prompt to save the log will be given after 99 records have accumulated in the default log, after which earlier records are progressively deleted so that the default log never holds more than 100 records.

Automatically update 'Current Security Bytes'

Check this box to have Xplor8 automatically update the Current Security Bytes whenever the bytes are changed, for example by writing new values to these locations. The default is checked. Whether or not this box is checked, Xplor8 continues to log all changes in the default security byte log.

Disable SB logging (use with caution)

Check this box to turn off the automatic logging of security bytes. The only circumstance in which I would consider this justified is if you are using Xplor8 to program a large run of chips with an identical program, in which case the repetition of identical automatic reads would clutter up the default log. See Security Bytes (page [14](#)) for more details.

7.7 Settings>Notes

This tab has a single memo edit box. Use this memo edit to make notes about the current project. These will be stored with the project file.

8 COMMANDS

8.1 General information

Commands to Xplor8 to carry out monitoring, debugging and loading operations are typed in to the Command Edit Box at the lower left-hand corner of the main form. When the Enter or Return key is pressed, the command is copied to the command history list in the window above the edit box. It does not matter where the cursor is in the command line when the Enter key is struck, provided that the command line edit box has focus. Each of the available commands is allocated a separate page in this manual with a full description of its use and limitations, and a summary of all the commands is given in Appendix C. Note also these points:

- If a syntax error is made in typing the command, Xplor8 echoes the command to the history window with the addition of an explanatory message. In the following example of a mis-typed List command, 008G is not a valid hexadecimal number:

```
L 0080 008G <-- Argument(s) not recognized
```

See the summary of Command Line Errors in [Appendix E](#) on page 79 for details of the possible messages.

- Previously issued commands may be recalled by using the up and down arrow keys, and then edited. Previous commands may also be reused by clicking or double-clicking on a line in the command history list. When commands are recalled, they are transferred to the command line edit box minus any comments, so that they may be corrected if necessary.
- Hitting the escape key clears the command line.
- The default number system for commands is hexadecimal (except for the specification of the number of repeats - see, for example, the [Unassemble](#) command, page 62). However, decimal and binary notation may also be used: precede a decimal number by a '#' symbol, and a binary number by '%'. There is no need to prefix hexadecimal numbers with the dollar sign, although it is not an error to do so. Leading zeros are optional. The following commands are identical:

```
L 0080 8F
L $0080 $008F
L #128 %10001111
```

- Commands may be typed in upper or lower case; it makes no difference as they are converted to upper case before being parsed. The following commands are identical:

```
L FFF0 FFFF
L fFf0 ffff
```

- Command elements may be separated by one or more spaces or by a single comma, or a mixture of the two. The following commands are identical:

```
L,0080,008F
L 80 8F
```

- Some commands can be limited to affecting only the local, Xplor8, copy of the MCU memory space (see [local memory](#) on page 11). All the sample L(List) commands above would result in listing MCU memory. Typing the following command lists only the local copy of memory:

```
LL 80 8F
```

The addition of the letter L after the first L limits the command to act only on the local memory. Because the local commands do not have to read and write to the MCU they will appear to work faster.

- Where a command expects a range of memory to be specified by a starting and an ending address, then it is always possible to replace the second (ending) address by a '+' sign followed by a value to add to the starting address. Thus the following two commands are identical in their action:

```
L 80 8F
L 80 +F
```

- Label names, as listed in a loaded symbol table, or as found in the register information file, may be used wherever an address is expected. Label names are partially case sensitive, see [Labels instead of Addresses](#) on page 42 for an explanation of how Xplor8 treats labels. The currently available symbolic labels may be viewed in the [Symbol Table/Register Display](#) (page 28).
- Certain commands conclude by pre-filling the command line with information so that the command may be repeated simply by pressing the <Enter> key. The following commands do this:

```
List Memory
Unassemble
Trace
Step Over
Go (Run)
```

In every case, only the first element of the command is written back to the command line. For example, if you wish to trace a program beginning at \$F800, and type: 'T3 F800' in order to trace the first three instructions, then after the command has executed, the command line will be filled in with 'T3', allowing you to trace the next three commands simply by pressing the <Enter> key.

8.2 Path Tokens

In order to save space in file name edit boxes, and on the command line, two common path names are presented in tokenized form:

- | | |
|-----------|---|
| [MyDocs]\ | This token replaces "My Documents\\"", itself an alias for a path such as: "C:\Documents and Settings\Current User\My Documents\" |
| [Xplor8]\ | This token replaces the installation path of the Xplor8 executable, e.g. "C:\Program Files\Xplor8\" |

The tokenized forms are also used within project files to store file information. This simplifies the backup and transfer of projects to another computer, since file information is interpreted locally instead of by using absolute paths.

8.3 Labels instead of Addresses

Label names, as listed in a loaded symbol table, or as found in the register information file, may be used wherever an address is expected. Label names are partially case sensitive.

Xplor8 looks up symbol names in an ordered fashion: first it checks for a name with exactly the case entered. If there is no case-sensitive match, then a case-insensitive search is done. If a single match is found, then this is used; however, if more than one label is found which matches the spelling (but not the case) of the label supplied, then the first matching spelling is used.

For example, suppose the following symbol file is loaded:

Main	8026
Dly1	802c
LongDly	8033
MAIN	8039
TestLoc	8042

- Entering 'Dly1' will find \$802C
- 'DLY1' will find \$802C (no other label has the spelling 'D L Y 1')
- 'Main' will uniquely find the value \$8026.
- 'MAIN' will uniquely find \$8039
- 'MaiN' will find \$8026 being the first instance of 'M A I N' in the list

The command:

```
L Main LongDly
```

would be the same as:

```
L 8026 8033
```

And the command:

```
F 8050 8050 TestLoc
```

would fill the two bytes at \$8050/1 with the address of TestLoc (\$8042)

All the control registers on an 'E' series chip could be listed by typing:

```
L PortA Config
```

this is equivalent to:

```
L 1000 103F (provided the registers are in their default location)
```

8.4 Clear Local Memory

CLM

Format:

CLM

This command only clears the [local memory](#) held within Xplor8. It does not affect the CPU or its memory at all.

8.5 Clear Output Window

CLS

Format:

CLS

This command only clears the output window. It does not affect the CPU or its memory at all.

8.6 Connect / Disconnect

Format

CONNECT | DISCONNECT

Command to open or close the RS232 serial COM port.

When disconnected, commands may still be given to Xplor8, but these will fail unless they affect only [local memory](#).

The 'Connect' speedbutton may be used as a quick way of issuing this command - see [Speedbuttons](#) on page 19.

8.7 Edit Flash Byte

EFB

Format:

EFB Address NewByte

Command to change a single byte of FLASH memory. 'Address' is an address in FLASH memory, 'NewByte' is the new value. If the value at Address is \$FF before the command, then the new value is simply written to memory; if not, then the whole page which includes the byte to be changed is read into Xplor8, the page erased, and then rewritten with the byte at 'Address' having the new value.

Example:

```
EFB EE00 1D
```

To change a single byte in random access memory (RAM or I/O registers) use the Modify Memory command.

8.8 Fill Memory

F

Format:

F StartAddress EndAddress|+Length ByteString|CharacterString

or

FL StartAddress EndAddress|+Length ByteString|CharacterString

To fill memory with a repeating sequence of bytes, the start address must be explicitly stated (an asterisk will not do). Either the end address, or a length must be specified, followed by the bytes or character string to be placed in memory. Character strings must be delimited by matching pairs of single or double quotation marks. The number of bytes or characters in the string is limited to 32. Byte strings must have two characters per byte.

The F form writes to MCU-controlled memory, FL writes only to [local memory](#).

Examples:

F 8000 800F 01	Fills MCU controlled memory from address \$8000 through \$800F with the byte \$01
F 8000 800F 414143	Fills MCU controlled memory from address \$8000 through \$800F with the byte sequence \$41, \$42 \$43 repeated as often as will fit.
FL 8000 +F 414243	As preceding example, but only fills Xplor8 local memory .
FL 8000 801F 'DEF'	Fills local memory from \$8000 to \$801F with the repeating sequence of bytes \$44, \$45, \$46.

To fill memory with a single instance of a byte string or character string, the start and end addresses may be entered as the same value. Memory is then filled with one copy of the byte string, or ASCII bytes corresponding to the given character string, starting at StartAddress; as in the following examples:

F 8000 8000 "Xplor8"	Fills MCU controlled memory from address \$8000 through \$8006 with the bytes \$4A, \$42, \$75, \$67, \$31, \$31.
F 8000 +0 'Xplor8'	Identical with previous example.
F 8000 +0 4A4275673131	Identical in result with the previous example (bytes specified directly, rather than as a character string)

The 'FL' form of this command may be used to fill memory anywhere in the 64 KB address space, but the 'F' form will fail if the destination is not covered by one of the address ranges specified in [Settings>Memory](#).

8.9 Find Bytes

FIND

Format:

FIND StartAddress EndAddress|+Length ByteString|CharacterString

or

FINDL StartAddress EndAddress|+Length ByteString|CharacterString

To find a string of bytes within memory, the start address must be explicitly stated (an asterisk will not do). Either the end address, or a length must be specified, followed by the bytes or character string to be found. Character strings must be delimited by matching pairs of single or double quotation marks. The number of bytes or characters in the string is limited to 32.

The FIND form searches MCU-controlled memory, FINDL searches only [local memory](#).

The address returned by this command is that of the start of the first occurrence of the string. Further occurrences of the same search string may be found using the NEXT command.

Examples:

<code>FIND 8000 800F 414143</code>	Searches MCU-controlled memory from address \$8000 through \$800F for the byte sequence \$41, \$42 \$43.
------------------------------------	--

<code>FIND 8000 +1F 'DEF'</code>	Same effect as preceding example.
----------------------------------	-----------------------------------

<code>findl 0 ff 39</code>	Searches local memory for the byte \$39 between addresses \$0000 and \$00FF
----------------------------	---

See also the description of the NEXT command.

8.10 Find Next

NEXT

Format:

NEXT

To find the next occurrence of a string of bytes previously specified with a FIND command. This command actually searches only [local memory](#), but this is unimportant as local memory will have been updated by the FIND command. No commands other than FIND or NEXT can be interposed between one use of FIND or NEXT and the NEXT command - for example, issuing the FIND command followed by the L(List) command will generate an error if the NEXT command is then issued.

The address returned by this command is that of the start of the next occurrence of the string.

For example, suppose that memory contained the following bytes:

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F	
8000	86	00	B7	10	08	86	38	B7	10	09	86	16	B7	10	28	B68.....(.
8010	10	28	8A	40	B7	10	28	44	45	46	44	45	46	44	45	46	.(.@(DEFDEFDEF
8020	A7	00	08	5A	26	FA	86	00	C6	07	CE	80	60	BD	80	86	...Z&.....'...

and that the command:

```
FIND 8000 802F 'DEF'
```

had found the first occurrence of the byte sequence \$41, \$42, \$43 at \$8017. Issuing the command

NEXT

would elicit the output:

```
String found at: $801A
```

and issuing NEXT again would produce:

```
String found at: $801D
```

8.11 Go (Run)

G

Format:

G [StartAddress]* [Breakpoint]]

Command to run a program on the MCU. If StartAddress is not specified, or an asterisk is used, then the current value of the 'T/G' address will be used. If StartAddress is specified, or an asterisk is used, then a [Breakpoint](#) (page 17) may be specified at which execution will halt. Programs may be started at an address in RAM or FLASH.

The user is responsible for ensuring that the breakpoint address is located at the first byte of an actual instruction. The [Unassemble](#) command (page 62) may help to locate this.

The breakpoint set as a result of issuing the G command will appear in black the Start/Break Point read-only list box. When the MCU program is executing but not yet halted at a breakpoint, the status bar will display the word 'Running'. When halted, the status display will change to 'Monitor Mode', and the breakpoint information will be highlighted in red, with the address of the next instruction to be executed on the line below.

Issuing the G command without any arguments will set the program running at the current value of the program counter. If no breakpoint is set (Break Register = 0000), there is no way for Xplor8 to regain control, unless the user's program terminates in a SWI instruction.

Issuing the G command with a single argument will set the program running at that address, without setting a new breakpoint.

If your program has stopped at a breakpoint, and you wish to continue to another breakpoint, type:

```
G * NextBreakAddr
```

where the asterisk is a placeholder for the current value of the program counter.

If you are stopped at a breakpoint and type 'G' alone, or type:

```
G * *
```

so as to continue on to break at the same address that you are starting from, Xplor8 first executes a private trace operation, then resets the breakpoint at your original starting address.

8.12 List Macros

LM

Format:

LM

Command to list the names of the macros in the currently loaded macro library file. See [Macros](#) on page 68 for more information on the structure of macro files and commands.

The following example lists the names of the macros in the currently loaded macro library file.

LM

If the file "Sample 1.mcr", supplied with the program, was the currently loaded macro library, then the above command would produce the following in the Output Window:

```
MACRO1          {Macro to process an S19 file}
MACRO2          {Trace testing macro}
```

The names of the individual macros are also copied to the Command History window, followed by an arrow thus:

```
MACRO1 <--
MACRO2 <--
```

This is so that any one of the macros may then be executed by double-clicking on its name in the Command History window (or by using the up/down arrow keys on the keyboard to select it)

8.13 List Memory

L

Format:

L [StartAddress]* [EndAddress|+Length]]
or
LL [StartAddress]* [EndAddress|+Length]]

Command to list memory. If no end address is specified, then 16 locations will be listed, beginning at StartAddress. If no start address is specified in the command, then 16 locations will be listed, beginning at the current value of the Xplor8 'L/U' address - see the [Information Sidebar](#) on page 21. The default value of 16 locations may be changed in [Settings>General](#).

The L form lists MCU-controlled memory, LL lists only [local memory](#).

Examples:

```
L 8000 8018      List MCU controlled memory from $8000 to $8008
L 8000 +18      Same as preceding example
```

The above two commands will produce something looking like this (of course the actual bytes in memory depend upon the programming):

```
      +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
8000  86 00 B7 10 08 86 38 B7 10 09 86 16 B7 10 28 B6 .....8.....(.
8010  10 28 8A 40 B7 10 28 7E 00                      .(.@..(~.
```

The command:

```
LL 8008
```

will list the 16 locations in local memory, starting at address \$8008, and finishing with location \$8017, for example:

```
      +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
8000                                10 09 86 16 B7 10 28 B6 .....(.
8010  10 28 8A 40 B7 10 28 7E                      .(.@..(~.
```

If the current value of the 'L/U' address were \$8010, then typing the following:

```
L
```

would list the 16 bytes of MCU controlled memory from \$8010 to \$801F. The value of the 'L/U' address is updated by the Trace and Go commands so that it has the value of the latest breakpoint, making it easier to see where the program has halted.

Format:

LD Path&Filename.rec|.s19

LDL Path&Filename.rec|.s19

or

LD Path&Filename.obj|bin StartAddress

LDL Path&Filename.obj|bin StartAddress

Command to fill memory from an S19 file or from a binary image file. The LD form writes to MCU-controlled memory, LDL writes only to [local memory](#).

A complete path and filename, with extension, must be supplied. If a file in S19 record format is to be loaded, then the extension must be '.s19' or '.rec'. If a binary image file is to be loaded, it must have the extension '.obj' or '.bin' and a starting address for loading must be specified.

The path part of the filename may be shortened by the use of tokens - see the section on [Path Tokens](#) on page 41. If the path and filename include spaces, then the whole argument string must be enclosed in single or double quotation marks, as in this example:

```
LD "C:\My Documents\HC908\S19 Files\First.S19"
```

The following example loads a binary image file into MCU controlled memory, starting at address \$9000:

```
LD "[Xplor8]\Samples\Bin Example 1.obj" 9000
```

Note the use of quotation marks to enclose an argument which contains spaces and would otherwise be wrongly interpreted.

To save having to type the file name and path, this command may be executed automatically from the Actions menu with one of the four items: Load S19 to MCU, Load Binary to MCU, Load S19 to Local and Load Binary to Local. The word 'LD' or 'LDL', as appropriate, followed by the full file name, is then written to the command edit box, and the command is executed. A speedbutton is provided for the common operation of loading an S19 format file to MCU-controlled memory.

The 'LDL' form of this command may be used to fill memory anywhere in the 64 KB address space, but the 'LD' form will fail if the destination is not covered by one of the address ranges specified in [Settings>Memory](#).

During the execution of the Load command, the Output window displays information on the areas of memory being written.

8.15 Mass Erase

ME

Format:

ME [Address]

Command to mass erase all of FLASH memory. 'Address' is optional; if it is not included, or has any value other than \$FFFF, then a confirmatory dialog appears to make sure that you really do wish to erase all the FLASH memory! By setting 'Address' to \$FFFF, this dialog is skipped, which is useful in macros.

This command is the only one which will work if the MCU is not unlocked by sending the correct security bytes.

8.16 Mode Switch

MODE

Format

MODE M

or

MODE U

Command to switch between monitor mode and user mode. Only available if the necessary hardware is available on the target board, and the correct options have been chosen on the Settings>COM Port tab. This command may also be issued using the 'Switch' speedbutton.

8.17 Modify Memory

M, MM

Format:

M StartAddress

or

MM StartAddress

Command to modify random access memory byte by byte. An explicit start address must be specified. When the command is given, Xplor8 will reply with the address and the current value at that address, followed by a period. Typing a new byte value, followed by the enter key, will change that memory location, then advance the memory address by one and present the next byte for modification.

Pressing the space bar followed by the enter key will preserve the existing memory value. Pressing the enter key alone will terminate the modify memory command. Typing a '+' followed by an offset (default hexadecimal) will advance the location to be modified by that amount, similarly a '-' will retard the location. Typing '=' followed by an address will make this address the new location for modification. An omitted offset defaults to 1, omitting the address defaults to the current address.

For example, typing

```
M 8000
```

might give

```
8000 34.
```

Completing this line thus:

```
8000 34.4A
```

and pressing the enter key will change memory location \$8000 from \$34 to \$4A and present the next memory location for modification thus

```
8001 C3.
```

If the enter key is now pressed without typing a new byte value, the command will terminate, leaving location \$8001 with the value \$C3. If the space bar was pressed before the enter key, then the value \$C3 would also have been preserved, but the command would not terminate, and the next location, \$8002 would be presented for modification. Typing:

```
8001 C3.+3
```

will advance the location to \$8004 without changing the byte at \$8001, and present the byte for modification:

```
8004 5F.
```

Typing an '=' and a new address moves modification to that address:

```
8004 5F.=8010
```

leads to:

8010 30.

This command cannot write to FLASH memory.

8.18 Page Erase

PE

Format:

PE StartAddress [EndAddr|+Length]

Command to erase one or more pages of FLASH memory. If only a single argument is supplied then only the page containing that address will be erased. If either StartAddress or EndAddress do not lie on an erase page boundary, a warning dialog appears to the effect that the range about to be erased is larger than you have specified. This dialog gives you the option to cancel the command.

Note that you do not need to explicitly erase memory before re-programming it; Xplor8 takes care of the necessary erasing as part of the Load Memory and Fill commands.

The page size depends upon the particular MCU that you are using.

8.19 Pause

PAUSE

Format:

PAUSE [milliseconds]

The Pause command has two forms. If PAUSE is followed by a number of milliseconds, then Xplor8 will pause macro execution for this length of time. 'Milliseconds' is a decimal number.

PAUSE is only available in macros, and following the G(Go) command.

If PAUSE is not followed by a number, then macro execution is paused until the user hits any key, or the character \$4B is received by the serial port. This lets the target MCU control the execution of Xplor8 macro commands. In this form of the command, the message 'Pause end signaled' appears in the Output Window when the \$4B character is received or the user hits a key. Note that the program will still be running on the MCU, and if there are further commands in the macro, these will then be executed.

Example:

PAUSE 1000 Suspends macro execution for 1 second.

8.20 Register Display and Change

R

Format:

R I/OReg|CtrlReg

or

R I/OReg|CtrlReg=NewValue

Command to display and modify the MCU Input/Output registers. I/OReg or CtrlReg stands for one of the HC908 I/O or control registers, e.g. PTA or DDRA. When using this command to change a register, no spaces should be left either side of the equals sign. The R command may be used in two different ways:

1. Issuing R followed by the name of an MCU I/O or control register displays the current state of that register, with the set bits highlighted in bold. For example, on the 908GP32:

R SPCR

might produce the following line in the output window:

```
0010 SPCR [$28,#40] SPRIE DMAS SPMSTR CPOL CPHA SPWOM SPE SPTIE
```

The output line shows: the address of the register, its name, its current value in hexadecimal, a comma, its current value in decimal, the eight bits named, and highlighted in bold if set.

Note that there is one refinement to this version of the command. If the register is one of those that have a hi and lo part effectively making up a sixteen bit register, then typing the name without the last H or L will display the current sixteen bit value, for example:

R TCNT

might produce:

```
100E TCNT [$5C4A , #23626]
```

2. Issuing R followed by the name of an MCU I/O register, an equals sign and a new value, will set that register to the new value (if the register is one that allows writing). Binary number representation is allowed also. Example:

R DDRB=03 or

R DDRB=%00000011

sets the DDRB1 and DDRB0 bits in the port B data direction register.

Note: This command can read, but cannot write, to registers which are implemented as FLASH locations.

8.21 Reset

RESET

Format

RESET

This command, which is only available when 'PC controls reset' is checked on the [Settings>COM Port](#) tab, and the necessary hardware is in place, allows Xplor8 to reset the MCU remotely by toggling an RS232 control pin. This is a convenience, but you can still use Xplor8 without this facility.

This command is only available when connected, i.e. when the COM port is open.

The 'Reset' [speedbutton](#) may be used as a quick way of issuing this command.

8.22 Save Memory

SV

Format:

```
SV StartAddress EndAddress|+Length Path&Filename.rec|.s19
SVL StartAddress EndAddress|+Length Path&Filename.rec|.s19
or
SV StartAddress EndAddress|+Length Path&Filename.obj|.bin
SVL StartAddress EndAddress|+Length Path&Filename.obj|.bin
```

Command to save a block of memory to a file in S19 or binary image format. The SV form saves MCU-controlled memory, SVL saves only [local memory](#).

The start address of the block to be saved must be stated explicitly (an asterisk will not do). Either the end address, or a length must then be specified, followed by the path and name of the file. The extension determines whether the data is saved in S19 format or as a binary object file.

The path part of the filename may be shortened by the use of tokens - see [Path Tokens](#) on page 41. If path and filename include spaces, then the whole argument string must be enclosed in single or double quotation marks.

The following example saves the memory from \$B600 to \$B6CE (inclusive) to the file Test1.S19:

```
SV B600 B6CE "[MyDocs]\68HC908\Test1.s19"
```

If successful, a 'Saving complete' message will appear in the Output Window:

To save having to type the file name and path, this command may be executed automatically from the Actions menu with one of the four items: Save MCU to S19, Save MCU to Binary, Save Local to S19, Save Local to Binary. The word 'SV' or 'SVL', as appropriate, followed by the full file name, is then written to the command edit box, and the command is executed. A speedbutton is provided for the common operation of saving MCU-controlled memory to an S19 file.

8.23 Send Bytes

SEND

Format:

Send BaudRate InterCharDelay ByteString

BaudRate	to suit the on-chip program. This baud rate applies only to the Send command, Xplor8 reverts to the MON08 interface communication rate at the end of the command. Note: not all baud rates are obtainable. On most PC systems the UART is a 16550 compatible device, and the Comms port driver is the standard one that comes with Windows. The result is that the only baud rates obtainable in practice are those which are integer divisions of 115200 baud, for example 7680 baud is obtainable because $7680 = 115200 \div 15$. If a rate somewhere between the available rates is entered in 'BaudRate', then the 16550 defaults to the first available rate above the rate entered, and a warning dialog appears.
InterCharDelay	a delay, in milliseconds, to be inserted between the sending of each character. A value of zero may be used. Short delays (<50 ms) are subject to some variability. The last byte sent is also followed by the inter-character delay.
ByteString	the bytes to be sent. Use pairs of hexadecimal characters, without any gaps, for example: 41420D0A.

SEND is a specialized command. It simply emits the specified bytes from the PC's serial port. Bytes sent are displayed in the output window in black; and any bytes received are displayed in red. Note that the half-duplex nature of the hardware interface on most target boards will cause each byte sent to be echoed once anyway. SEND is one of the few commands available in user mode.

Examples

The following sends the ASCII characters 'ABC' at 9600 baud with no delay between each:

```
SEND 9600 0 414243
```

The following example sends the bytes necessary to read a single memory location. \$4A is the MON08 command to read a single memory location, \$FF81 is the location being read:

```
send 9600 20 4AFF81
```

The command replies with: (the \$D7 depends, of course, on what was in \$FF81)

Sent/Received: 4A4A4AFFFFF818181D7

Format:

TERM

This command opens the [Terminal Window](#) (see page 74). It is included so that a macro can put the user in direct terminal communication with a target board. Suppose, for example, that the demonstration program included in the distribution, TermDemo.asm, has been compiled to run at \$8000. Then the following macro might be used to load the demonstration program, start it running and open the terminal window, whenever the MCU is re-booted:

```
DEFM Autostart
BEGIN
    LD "C:\Program Files\Xplor8\Samples\TermDemo.s19"
    G 8000
    TERM
END
```

Format:

T[Repeats] [StartAddress]*]

Command to trace a program in MCU controlled memory. If no explicit start address is specified, then tracing will begin at the current value of the 'T/G' address. Repeats is an optional number which specifies how many times the command is to be repeated. Note that repeats is specified as a decimal number, unlike all other values which are entered in hexadecimal. The value of 'Repeats' follows the letter T without a gap. When the command is given, Xplor8 carries out the following operations:

1. The instruction at the starting address is disassembled to discover the address of the following instruction (or instructions if there is the possibility of a branch);
2. Xplor8 determines which is the effective following instruction, based on the opcode, the contents of the CCR, and on other information as appropriate.
3. A breakpoint is then written to MCU memory to 'catch' the program after execution of the current instruction. This breakpoint is displayed in the [Start/Break Point](#) window (page 22);
4. The MCU is set running at the current instruction, the status line is updated to read 'Tracing', and the instruction at the trace starting address appears in the breakpoint display in green;
5. As soon as the MCU stops at a breakpoint, that breakpoint is highlighted in red in the breakpoint display, the breakpoint information is copied to the output window, and the status line reverts to: 'Monitor Mode'. If populated, the watch window display is also updated.
6. The above processes are repeated if 'Repeats' is specified as greater than 1.

The Xplor8 internal trace starting address, T/G, and the default list and unassemble address, L/U, are also updated to show where the last trace instruction has stopped; so issuing the 'T' command alone will continue tracing from where the previous trace operation halted.

Care needs to be taken if tracing is begun at an RTS or RTI instruction if program execution has not arrived at that point by itself, e.g. as the result of issuing a [Go \(Run\)](#) command (page 48). This is so because the next address at which tracing (or stepping over - see the [Step Over](#) command on page 61) will halt is computed from the preceding value on the stack: this value will be quite arbitrary if tracing is begun 'out of the blue' at either of these instructions.

Examples:

T 800B	Begins tracing with the instruction at address \$800B
T3	Begins tracing at the current value of 'T/G' address and traces the next three instructions

8.26 Step Over

O

Format:

O[Repeats] [StartAddress]*]

Command to step over a program in MCU controlled memory, that is to jump over subroutine calls. If the next instruction is not a subroutine call, then this command behaves exactly like the [Trace](#) command. The 'T' and 'O' commands may be freely inter-mixed. If no explicit start address is specified, then stepping/tracing will begin at the current value of the 'T/G' address. When the 'O' command is given, Xplor8 carries out exactly the same five operations listed under the 'T' command, with the exception that the 'following instruction' mentioned in operation no.2 will be adjusted in the case of subroutine calls to be the next sequential instruction rather than the instruction at the entry to the subroutine.

'Repeats' is an optional number which specifies how many times the command is to be repeated. Note that repeats is specified as a decimal number, unlike all other values which are entered in hexadecimal. The value of 'Repeats' follows the letter O without a gap.

The T/G address (see [Information Sidebar](#) on page 21) is also updated to show where the last trace instruction has stopped; so issuing the 'O' command alone will continue stepping from where the previous trace or step operation halted.

Supposing that the following fragments of code are in memory:

E00E	LDA	, X
E00F	INC	\$01, SP
E012 DLYLOOP	BSR	DELAY
E014 FINISH	PULA	
E015	SWI	
E016	NOP	
E017 DELAY	LDA	#\$02
E019 DLY1	DECA	
E01A	BNE	DLY1
E01C	RTS	

Issuing the command:

T E00E

will begin tracing with the instruction at address \$E00E, and will halt at \$E00F. Issuing either 'T' or 'O' by itself will then advance tracing by one instruction, halting at \$E012. Issuing 'T' alone would then trace into the subroutine, and halt at \$F017, whereas issuing the 'O' command would step over the subroutine (the subroutine is still executed), halting at \$E014.

Note also the caveat in the [[~JUMP Trace](#)] command concerning beginning tracing or stepping at an RTS or RTI instruction.

8.27 Unassemble

U

Formats:

U[Repeats] [StartAddress]*

or

UL[Repeats] [StartAddress]*

or

U [StartAddress]* [EndAddress]+Length

or

UL [StartAddress]* [EndAddress]+Length

Command to unassemble (disassemble or reverse engineer) memory. If no start address is specified, or an asterisk is used, Xplor8 will use the current value of its own starting address, in L/U. Either an ending address, or a number of repeats may be specified (but not both). Repeats is an optional number which specifies how many times the command is to be repeated. Note that repeats is specified as a decimal number, unlike other values which are entered in hexadecimal. The value of 'Repeats' follows the letters U or UL without a gap. Code cannot be unassembled between \$FFFC and \$FFFF.

The U form unassembles MCU-controlled memory, UL unassembles only [local memory](#).

The following example disassembles the 10 instructions beginning at \$F800:

u10 f800

This might produce something like the following output:

0001:	F800	C6FFC0	LDA	\$FFC0
0002:	F803	A1FF	CMP	#\$FF
0003:	F805	2703	BEQ	\$F80A
0004:	F807	C7FC00	STA	\$FC00
0005:	F80A	6E091F	MOV	#\$09,\$001F
0006:	F80D	6E081E	MOV	#\$08,\$001E
0007:	F810	450C80	LDHX	#\$0C80
0008:	F813	5BFE	DBNZX	\$F813
0009:	F815	6E0236	MOV	#\$02,\$0036
0010:	F818	3F80	CLR	\$0080

The information on each line is: line number, address, bytes at that address, instruction, operands. In certain instances a comment will be added, for example if an unrecognized opcode is encountered. A symbolic disassembly is also possible - see Using a Symbol Table below.

The above result would also be achieved by issuing the command:

UL 0 +17

To reverse engineer a program available as an S19 or binary image file, it is not necessary to connect to an MCU at all; the program may simply be loaded into local memory at the correct starting address, and the 'UL' command used to disassemble it.

The readability of disassembly listings may be enhanced by checking 'UnAsm: add spaces at code breaks' on the [Settings>Debug](#) tab. This will add a blank line in unassembled listings after the following instructions:

- Unconditional jumps, including the BRA (Branch Always) instruction. No space is left if the jump address is that of the immediately following instruction, i.e. the jump was included simply to use up processor cycles.
- Return from Interrupt, RTI
- Return from Subroutine, RTS

8.28 Using a Symbol Table

A symbol file may be nominated to Xplor8 so that when addresses are encountered during disassembly which match a symbol value, then the symbol is displayed instead of the address. The symbol file is a plain text file. Xplor8 will cope with any file, provided that:

1. The symbol name appears first on the line;
2. The symbol value appears next, and is separated from the name by at least one space, comma or tab character. The value must be in hexadecimal but it is optional for it to be preceded by a '\$' sign
3. After the value any characters will be ignored, for example: comments, occurrences of the symbol, etc. (provided they are separated from the value by at least one space, comma or tab character)
4. Lines which cannot be interpreted as being a name followed by a value are ignored. This means that most heading lines will be discounted.
5. Some assemblers introduce the ASCII form feed character, \$0C, at the top of the listing - all control characters are ignored.

To make use a symbol file, select one of the radio buttons on the [Settings>Debug](#) tab. If the nominated file cannot be found, or cannot be interpreted, no symbol table will be loaded. The example given in the description of the [Unassemble](#) command (page 62) might look like this if a symbol table was in use:

```
0001:  F800 RomStart      C6FFC0          LDA  $FFC0
0002:  F803                A1FF           CMP  #$FF
0003:  F805                2703           BEQ  No_Trim
0004:  F807                C7FC00          STA  $FC00
0005:  F80A No_Trim        6E091F          MOV  #$09,CONFIG1
0006:  F80D                6E081E          MOV  #$08,CONFIG2
0007:  F810                450C80          LDHX #$0C80
0008:  F813 Delay          5BFE           DBNZX Delay
0009:  F815                6E0236          MOV  #$02,OSCSTAT
0010:  F818                3F80           CLR  RamStart
```

One or two factors have to be borne in mind when using a symbol table:

- Labels are partially case-sensitive. When looking-up the symbol table, Xplor8 searches first for an exact case match, and returns the associated value if found, then makes a search ignoring case. This allows you to have two labels with the same spelling, but different capitalization (if your assembler can do it).
- In an assembly language program, more than one label may reference the same value. Xplor8 has no way of knowing which label is the correct one if such a value is encountered - it simply picks the first label in the list which matches the given value.
- Labels longer than 8 characters will cause the output listing to become mis-aligned.
- Only 16-bit values are converted to labels, except in the case of the single byte representing an address in page-zero memory when a Direct Address mode opcode is encountered.

The currently-loaded symbol file may be viewed by clicking on 'Symbol Table' in the View menu, or using the keyboard shortcut: Ctrl+L

Format:

UNLOCK [SecurityBytes]

Use the unlock command to send the eight security bytes to unlock FLASH memory. 'SecurityBytes' is an optional string of eight bytes in hexadecimal notation; if omitted, the command will send the current security bytes nominated in Settings>Security Bytes ([page 38](#)). The format of the security byte string is unimportant; Xplor8 strips out spaces before sending, so the following are the same:

```
Unlock F895FFFFFFF895F895
UNLOCK F8 95 ff FF f895      f895
```

When the bytes are sent, confirmation appears in the output window thus:

```
Unlocking bytes echoed: F895 FFFF F895 F895
```

These are the bytes as echoed by the MCU monitor-entry firmware.

If the command succeeds, the following message will appear in the output window, and the second panel of the status bar will read 'FLASH unlocked':

```
Success! FLASH unlocked.
```

If the command fails, a message similar to the following will appear in the output window, and the status panel will read: 'FLASH locked' (the address \$0080 may change depending on the particular MCU you are using):

```
Failure - bit 6 of $0080 not set
```

If the bytes are sent, but the MCU fails to respond by sending a 'break', then a time-out message will appear. Have you remembered to power up the target board?. Some Windows COM port drivers appear to have trouble recognizing the breaks issued by the MON08 firmware see [Alternative 'Break' Detection \(page \)](#).

This command is available when connected, i.e. when the COM port is open. The 'Unlock' button may be used as a quick way of issuing this command - see [Speedbuttons, page 19](#).

8.30 Verify

V

Format:

V Path&Filename.rec|.s19

or

VL Path&Filename.rec|.s19

Command to verify memory by comparing it with an S19 format file. The V form verifies MCU-controlled memory, VL verifies only [local memory](#). If the path and filename includes spaces, then the whole argument string must be enclosed in single or double quotation marks, as in this example:

```
V "C:\My Documents\HC908\S19 Files\First.S19"
```

The path part of the filename may be shortened by the use of tokens - see [Path Tokens](#) on page 41. This command may also be activated from the 'Actions' menu.

If the command is successful, the message 'MCU memory verifies OK' will appear in the output window. If the memory does not verify correctly, then a message similar to this will appear:

```
Verify fails at 8043      MCU = 8D      S19 = 00
```

As another example of the use of the Verify command, the following macro might be used to program and verify FLASH:

```
DEFM Load_Flash
BEGIN
    LD "[MyDocs]\Mot 08\My Program 1.S19"
    V "[MyDocs]\Mot 08\My Program 1.S19"
END
```

The Xplor8 Verify command strictly compares the S19 file information with the bytes in memory at the addresses specified in the S19 file.

8.31 Verify Erase

VE

Format:

VE StartAddress [EndAddress|+Length]

Command to verify that MCU memory is erased (bytes = \$FF). If no arguments are supplied then the whole of FLASH memory will be checked. If StartAddress only is specified, then only that one address will be checked. Specifying both StartAddress and EndAddress will check the given range. The command works by loading a utility file to RAM, not by reading in MCU memory, so it works fast. Verification always begins at StartAddress, but the Verify Erase command is the only one which allows EndAddress to be less than StartAddress; so memory may be verified in a downward direction.

If both arguments are omitted, VE checks the whole of FLASH memory. If any memory is found to contain code, then the output window returns information on the occupied blocks. A block is here taken to mean a range that is an exact multiple of the erase page size.

For example, if the command:

```
ve F7f0 F7FF
```

were issued, then the program might reply with the following message in the Output window, showing that memory over this range is fully erased:

```
Memory erased over range F7F0 to F7FF
```

While the command:

```
VE F7F0 F800
```

might produce:

```
Memory NOT fully erased. At F800 MCU memory = 59
```

The command without any arguments thus:

```
VE
```

might produce

```
Memory NOT fully erased.
```

```
Blocks occupied at:
```

```
EE00..EE3F
```

```
F800..F8BF
```

```
FFC0..FFFF
```

9 AUTOMATION

9.1 Introduction

It is possible to automate the action of Xplor8 in four ways:

1. Three commonly-used commands can be issued automatically at start-up:
 - Connect / Disconnect
 - Reset
 - UnlockTo use this feature, tick the appropriate boxes in [Settings>General](#).
2. A group of commands may be executed automatically immediately after the initial commands mentioned above - see [Boot Script](#) on page 69.
3. A group of commands in a macro script named 'AUTOSTART' may be executed automatically after booting, see [Autostart Macro](#) (page 70).
4. Any number of commands may be grouped into named macros, which can then be executed by typing their names into the command line, or selecting one from a drop-down list.

See the section on [Macros](#) below for an explanation of macro scripts. Xplor8 includes a simple [Macro Editor](#) for writing macro script files, or macros can be 'learnt' as you type in commands.

9.2 Macros

Macros allow a succession of commands, grouped together under a named macro heading, to be executed sequentially, as though they were typed successively on the command line. This saves a lot of repetitive typing. Macros are stored as plain text in Macro Library Files. Each library file can contain an unlimited number of separate macros, and each macro can contain any number of statements. Macro library files have the extension '.mcr', and a typical one might look like this:

```
* Sample Macro library file (Sample1.mcr)
*
DEFM MACRO1           {Macro to process an S19 file}
BEGIN
    LD "[Xplor8]\Samples\Looptest.S19"
    L 8000 +30
    U6    8000
END
DEFM MACRO2           {Trace testing macro}
BEGIN
    F 8000 +3F 01
    LD "[Xplor8]\Samples\Looptest.S19"
    L 8000 +3F
    T 8000              {Trace}
    T                   ; NOTE: comments may also begin
    T                   // with a semicolon or 2 slashes
    S
    L 8000 +3F
END
```

Definition of a macro begins with the word 'DEFM' (Define Macro) followed by the name of the macro. The name may in turn be followed by up to 10 [Replaceable Parameters](#) (see page 70) . The next line must be 'BEGIN'. Thereafter the commands that make up the macro are written in the order in which they are to be performed. The last line of the macro must be 'END'.

You should note the following:

- Blank lines, and those beginning with an asterisk in the first character position, are ignored.
- The macro name following DEFM cannot contain spaces or commas.
- If one of the following is encountered on a command line:
 - an opening curly brace (the closing curly brace is merely treated as part of the comment),
 - a semicolon, or
 - a double forward slash.then everything that follows on that line will be treated as a comment.
- Upper and lower case may be freely mixed.
- Spaces, tabs or commas may be used as separators.

Macros may call other macros, but should not call themselves. When processing a command, Xplor8 checks the command against the list of macro names first, before checking whether the command is one of the built-in ones. This has the side effect that if a macro is given the same name as one of Xplor8's native commands, it will supersede the native command, which then becomes inaccessible.

9.3 Boot Script

Command script to run on start up, and before any [Autostart Macro](#) (See following section). Any number of Xplor8 commands may be used, including any macro in the current macro library. This script is entered on the [Settings>Macros](#) tab. It is independent of any macro script nominated elsewhere and is stored with the project data, not in a separate file. This script may be arranged to run after:

- Connecting
- Resetting
- Unlocking
- or not at all

Do **not** include the 'DEFM', 'BEGIN' and 'END' pseudo-commands which are used in macro library files - see Macros above.

A typical use might be to alter the Flash Block Protect Register after unlocking (where it is implemented as a RAM location) so that a file may be loaded to FLASH. If the following line is typed into this box:

```
R FLBPR=FF
```

then FF will be written to FLBPR immediately after unlocking - see [Register Display and Change](#) (page 55) for more details of the 'R' command.

9.4 Autostart Macro

If a macro named 'AUTOSTART' is part of a library file already loaded when the MCU is re-booted, then this macro will be played automatically after the FLASH is unlocked, and after any boot script. Even where a macro library file does contain an 'Autostart' macro, the automatic playing may be suppressed by checking the 'Disable Autostart macro (if any)' box on the [Settings>Macros](#) tab.

Here is an example of an auto start macro which will un-protect FLASH memory, write an S19 file to memory and start a program executing on a 908JB8 chip:

```
DEFM Autostart                ; Macro to load and execute a file
begin
    R FLBPR=FF                ; Un-protect FLASH
    LD "[MyDocs]\Asm\Test.S19" ; Load a file
    G DC00                    ; begin execution
end
```

The need for an 'autostart' macro has been partially obviated in Xplor8 Version 5 by the addition of the [Boot Script](#) facility (see page 69).

9.5 Replaceable Parameters

Macros may contain replaceable parameters. Up to 10 parameters may be specified, from @0 to @9 or %0 to %9. @0 is the same parameter as %0, and reference to one is a reference to the other. Following the name of the macro on the DEFM (define Macro) line should be a list of the parameters that will be used in the macro.

Replaceable parameters may be defined (on the DEFM line) in any order, for example:

```
DEFM MyMacro @3 @5 @1
```

is valid. But note that when the macro is called as a command, then it will require three arguments, and these will be allocated in strict left-to-right order. So, calling MyMacro thus:

```
MYMACRO 8000 8001 8002
```

will assign \$8000 to @3, \$8001 to @5 and \$8002 to @1.

In the following (over-complicated) example MACRO3 is called by MACRO4, with one replaceable parameter:

```
DEFM MACRO3 @0                ; macro takes one replaceable parameter
BEGIN
    LD @0                      ; (LD %0 would do the same thing)
END
DEFM MACRO4
```

```
BEGIN
    MACRO3 "[Xplor8]\Samples\Looptest.S19"
END
```

Playing MACRO4 in the above example thus:

```
Macro4
```

is equivalent to:

```
LD "[Xplor8]\Samples\Looptest.S19"
```

9.6 Playing and Recording Macros

9.6.1 Playing

Macros may be played by:

1. typing the macro name as a command on the command line, or
2. selecting a macro from the drop-down list adjacent to the 'Play' speedbutton, or
3. selecting a macro from the 'Play' item in the 'Macro' menu, or
4. clicking on the 'Play' speedbutton (provided a macro has been previously selected).

Only method 1. above allows the playing of macros that have [Replaceable Parameters](#) (see above), since these must be added after the name.

For example, to play the macro MACRO1 listed in the [Macros](#) section above, the following command would be issued:

```
MACRO1
```

As the macro plays, each line of the macro is copied to the command edit box and executed, just as if it had been typed in by hand and the enter key pressed.

Macros may also be played via the 'Macros' menu and via the 'Play Macro' speedbutton.

Xplor8 may be configured automatically to stop playing a macro under certain conditions:

- If the macro contains an incorrectly formatted command, that is one that results in an error message beginning with the '<--' characters, or
- The command is one that can have an unsuccessful result, e.g. a Verify command might result in memory not being verified correctly.

These configurations are selected in [Settings>Macros](#).

9.6.2 Recording

Macros may be recorded, i.e. the commands typed into the command edit box will be added to a macro script. This is often referred to in other programs as 'learning' a macro. Access this via the Macro menu

or the speedbuttons. Recorded command lines are appended to the end of whatever library script is open in the macro editor.

9.6.3 Stopping Playing or Recording

Use the 'Stop' speedbutton, or click the 'Record' or 'Play' button again, as appropriate, when it will return to its inactive state. While the command edit box has focus, hitting the <escape> key will also stop playing and recording.

9.7 Macro Editor

Xplor8 includes a simple text editor for macros which may be launched in three ways:

- With the 'Edit Macro' speedbutton, or
- From the 'View' menu on the main form, or
- By using the shortcut Ctrl+E,

The editor's size and position will be remembered in the Windows registry on closure.

If changes have been made in the editor, but not saved, Xplor8 will prompt the user to save the changes before finally closing down.

Macro Editor Menu Items

File

New	Clears the macro editor in preparation for writing or recording new macros.
Open ...	Opens a text file into the editor. The default file extension is “.mcr”. The shortcut key is Ctrl+O.
Save ...	Saves the contents of the macro editor to a text file, default extension .mcr. The shortcut key is Ctrl+S.
Save As...	Saves the contents of the macro editor to a text file with a new name.
Close	Closes the macro editor.

Edit

Undo	Undoes the previous change. Shortcut: Ctrl+Z
Cut	Cut the current selection to the Windows clipboard. Shortcut: Ctrl+X
Copy	Copy the current selection to the Windows clipboard. Shortcut: Ctrl+C
Paste	Paste the current contents of the Windows clipboard into the editor at the current cursor position. Shortcut: Ctrl+V
Select All	Select all the text in the editor. Shortcut: Ctrl+A

Delete Line	Delete the line containing the insertion point. Shortcut: Ctrl+Y
Make into Macro	If a line or lines have been selected in the editor, then this menu item will insert the necessary 'DEFM', 'BEGIN' and 'END' lines to form a complete macro definition. There is no need to be precise about the selection, provided that the selected area begins somewhere in the first desired line and ends somewhere in the last desired line.
Check Syntax	Checks that 'DEFM', 'BEGIN' and 'END' lines appear in the right order and that the rules on replaceable parameters have been followed. It does not check that the commands forming the body of the macro are legal.

Macro

Play	Select a macro to play from the sub-menu.
Record	Begin recording. Commands typed into the command edit box are copied and appended to the text in the macro editor.
Stop	Stop recording or playing.

Options

Keep on Top	Click this menu item to keep the macro editor form on top
Font Size	Select a font size for the macro editor from the sub-menu

Help

Index	Opens Help Topics at the Index tab. Shortcut is F1.
Contents	Opens Help Topics at the Contents tab. Shortcut is Shift+F1.
Help on Macro Editor	Opens the 'Macro Editor' help topic.

Note:

The Edit menu is also duplicated as a right-click context menu while you are working in the editor. A single line may be executed by double clicking it; in which case it is transferred to the command edit box for execution.

10 TERMINAL WINDOW

10.1 Introduction

Xplor8 has an integrated terminal window which allows the user to send and receive bytes with a connected MCU, provided a suitable program is running on the MCU. Bytes may be sent as ASCII text, or in their hexadecimal representation. A demonstration HC908 program is included in the distribution file; this is in assembly language and will need compiling to suit the memory architecture of the proposed target MCU - see the section on the [Terminal Demonstration Program](#) below.

The terminal window may be activated by:

- clicking the item in the View menu of the main form,
- typing the keyboard shortcut Ctrl+T, or
- issuing the command 'Term' (see [Launch Terminal Window](#)) on the command line.

10.2 Window Layout and Features

The window is divided into two panes. The left hand pane displays bytes sent (black) and received (red) in their alphabetic representation as far as possible. Non-printing characters display as hollow rectangles (this option may be suppressed - see [Terminal Window Settings](#) below). The right hand pane displays the same characters in their hexadecimal representation, as pairs of numbers in the range 00 to FF.

The division between the two panes is a moveable splitter bar, by default the right hand pane is twice as wide as the left hand one because the hex representation of bytes takes twice the width of their ASCII form.

Bytes may be entered in either pane at will. The <Tab> key will swap between panes. The currently active pane is highlighted in aqua. Only printable characters can be entered in the left hand pane.

The <Backspace> key operates differently in the two panes. On the left, it erases the on-screen characters and, if 'Send Immediately' is in force, also sends the backspace character (\$08) to the MCU. On the right it either erases the first character of a part-complete byte, or both characters of a complete one, but in no case does it transmit the \$08 byte.

The action of the <Enter> key depends upon the terminal settings. The terminal can be configured either to send bytes immediately they are typed, or only upon pressing <Enter>. In the latter case, additional bytes can, optionally, be sent when <Enter> is pressed. These bytes may be chosen by the user.

Word wrapping may be enabled in the two panes, from the View' menu.

The terminal window uses the communication baud rate selected in 'Baud Rates' on the [Settings>COM Port](#) tab, and the COM port must be connected (opened) before the terminal can be used.

10.3 Interaction with the Monitoring/Debugging Functions

It is basically only possible to use either the main Xplor8 monitor program or the terminal window at any one time, although it is quite possible to switch from one to the other provided the correct sequence is followed. A study of the section on the demonstration terminal communication program provided in the distribution, and of the code in that program, will answer most questions about the mutual compatibility of the monitor and terminal.

10.4 Terminal Window Menu Items

File

Logging	Select 'On' or 'Off' from the sub-menu. If logging is on then an internal log is kept of the bytes sent and received by the terminal window, and this log is appended to the file specified in Terminal Window Settings (see below) whenever the terminal session is completed. If no file has been specified, then this menu item is unavailable.
---------	--

Send Image File	Opens a File Open dialog for the user to select a file to be sent to the MCU as though it had been typed in at the terminal window. Any file nominated here is treated as a file of bytes which are sent sequentially. If 'Local Echo' (see below) is on, then the characters from the file will be echoed in the terminal windows as the file is sent, whether or not the MCU is itself echoing bytes received. Note that no handshaking is implemented, so the MCU has to be able to process the received bytes at the current communication baud rate, plus the nominated inter-character delay (see Terminal Window Settings). Keyboard shortcut is Ctrl+I
-----------------	--

Notes:

1. Files larger than 64 KB will not be sent.
2. Shortcut is Ctrl+I.
3. Sending of a file may be aborted by pressing the <Escape> key.

Close	Closes the terminal window and returns to the main form. Shortcut key is Ctrl+Q.
-------	--

Edit

Clear Windows	Clears the two panes
---------------	----------------------

View

Default Layout	Provides a default size and position for the terminal window. This may be used as a starting point for the user's preferred size and position; whatever is subsequently chosen will be remembered in the Windows Registry until next time.
----------------	--

Keep on Top	Checking this menu item will force the terminal window to stay on top of other forms even when inactive.
-------------	--

Font Size	Allows the user to chose the font size for the Ascii and Hex edit windows. The as-supplied default size is 8pt.
-----------	---

Word Wrap Check to wrap text in the display.

Settings

Terminal Settings... Click to open the Terminal Window Settings dialog.

Help

Index Opens Help Topics at the Index tab. Shortcut is F1.

Contents Opens Help Topics at the Contents tab. Shortcut is Shift+F1.

Help on Terminal Opens the 'Terminal Window' help topic.

10.5 Terminal Window Settings

Dialog for customization of the Terminal Window. Access this dialog from the 'Settings' menu item.

Show non-displayable characters as hollow rectangles

Tick this check box to cause non-displayable Ascii codes, those from \$00 to \$1F and from \$7F to \$FF, to display as a hollow rectangle in the left hand, ASCII, pane. When this box is not checked, no character is displayed at all. The default is to have these characters displayed as hollow rectangles.

Echo sent characters locally

Tick this check box to have characters typed at the keyboard echoed locally before being sent.

Send only on <Enter>

If this box is checked, then bytes typed into the window will not be sent until the <Enter> key is pressed. If unchecked, bytes are sent immediately they are typed.

Bytes to send when <Enter> is pressed

Enter in the combo box any byte sequence that it might be desirable to send, in addition to whatever has been typed, when the <Enter> key is pressed and 'Send only on <Enter>' is checked.

Bytes received which trigger a new line in the terminal windows

Enter in the combo box a byte sequence which, when received, will cause a newline in the terminal window.

Bytes received which trigger closure of the terminal window

Enter in the combo box a byte sequence which, when received, will cause focus to return to the main form. This is provided to speed up switching between the terminal and the de-bugging facilities.

Log File

Specify a name in the edit box, or choose a filename in the 'Open...' dialog, a file to become the terminal log file.

Delay (in milliseconds) between characters when sending a file

Enter a time in milliseconds which will be inserted between the sending of each character from a binary image file, or from the keyboard when 'Send only on <Enter>' is checked. Valid values are 0 ms to 100 ms.

10.6 Terminal Demonstration Program

In the distribution file is the program 'TermDemo.asm', which is installed by default in the '..\Samples\ sub-directory. This is designed to allow a quick demonstration of the capabilities of the terminal window. It will be necessary to assemble it to suit the available memory of the target MCU - instructions are provided in the heading comments of the file. After assembly, the program may be loaded with

```
LD "C:\...your path...\TermDemo.s19"
```

Execute a [Go \(Run\)](#) command (page 48) to the start of the terminal program body:

```
G xxxx
```

Now, without issuing any further Xplor8 commands, type Ctrl+T or use the [View](#) menu to launch the terminal window. In [Terminal Window Settings](#) (page 74), make sure that the check box 'Send only on <Enter>' is ticked, and that the three edit boxes are blank

Typing any character in the left hand pane, or byte in the right hand one, and pressing <Enter> should produce an echo of the same byte from the program running on the MCU. Typing 'Hello' should produce the echo 'world'. Experiment with immediate sending by un-ticking 'Send only on <Enter>' in Terminal Window Settings. To exit, type 'Exit' (capital E, lowercase x, i & t). After send 'Exit' the program executes an SWI instruction in order to return to monitor mode.

If the following byte string is entered in the combo box 'Bytes received which trigger closure of the terminal window' in Terminal Window Settings (or is selected from the drop-down list):

```
45786974 (these are the ASCII values for "Exit")
```

then typing 'Exit' will automatically transfer control back to Xplor8 and de-activate the terminal window.

11 ERRORS

11.1 Communication and Echo Errors

When Xplor8 sends a byte to the monitor firmware it usually expects to receive an echoed byte. By comparing the transmitted and received bytes, the program is able to assure the integrity of the RS232 communications link. If the echo is wrong, Xplor8 shows an error dialog. Two classes of byte error may be distinguished: Errors in the command byte received by the monitor firmware and errors in bytes being written to memory. These are referred to, for convenience, as ‘Comms Errors’ and ‘Echo Errors’.

11.1.1 Comms Error

This occurs when Xplor8 loses synchronisation with the monitor firmware on board the MCU. The command byte sent to the MCU to initiate a command has been wrongly echoed back to Xplor8. This usually happens when:

- The stack has been accidentally overwritten, or
- an operation has left the MCU in an unknown state.

It is usually necessary to reset the MCU and re-enter monitor mode when this happens. Where [remote reset](#) is available, the dialog appears with [Yes] and [No] buttons, allowing you to click [Yes] for an immediate re-boot.

When this error occurs, Xplor8 writes diagnostic information to the Output Window - See [Diagnostics](#) on page 79.

11.1.2 Echo Error

This occurs when Xplor8 is sending data and the MCU target echoes back a different character from the one that was sent. This happens when:

- the memory at an address could not be altered. It might, for example, be a location in ROM, or a location in FLASH when the bits in the FLBPR register do not allow writing to FLASH, or
- the memory at an address is a control register which has one or more read-only bytes, or

It should not be necessary to reset the MCU after this error as communication has not been lost.

Note that this error can occur part way through the execution of a command, so that some memory may have been correctly written, while some may not be written at all.

When this error occurs, Xplor8 writes diagnostic information to the Output Window - See [Diagnostics](#) below.

11.2 Diagnostics

When an error dialog is displayed, diagnostic information is added to the Output Window.

In the following example, I added the line 7FF0..7FFF to the 'External RAM' box in [Settings>Memory](#). Now this area of memory is unimplemented on my system, so trying to use the [Fill Memory](#) command will fail:

```
F 7FF0 7FF0 12
```

produces:

```
Echo error writing RAM
  Addr 7FF0  Sent 12  Rcvd BE
  State:Stopped
  Reboot MCU? = No
```

The constants in the information above are:

Addr	Address of byte at which the echo occurred. In the case of a Comms error writing the CPU inherent registers, an address as such is meaningless, and will always be zero.
Sent	Byte sent from the PC to the monitor firmware (MON08)
Rcvd	Byte as echoed by the monitor firmware (MON08)
State:	The information in the left-hand panel of the status line.

The line 'Resett MCU? = Yes|No' will only appear when the [remote reset](#) option is in use.

11.3 Error Report

Use this dialog (open in [Help](#) menu) to prepare a plain text report if you need to send information by email with a request for support. Fill in the edit boxes as you feel necessary, then press 'Make Text File...'. This will make a plain text file called something like ErrRpt_20060706_1820.txt, or whatever is the current date and time. Then send it to me at john.beatty@virgin.net

11.4 Command Line Errors

These are the error messages generated by Xplor8 when it cannot carry out a command. They appear in the Command History box, following a command in error and beginning with the symbol '<--'. From the right-click context menu in the Command History Window you can call up the relevant help topic on one of these errors. For a printable list of all the command line errors, see Appendix E of the Manual.

APPENDIX A - ACKNOWLEDGMENTS

Motorola/Freescale

For their flexible design of the HC908 micro controllers and for their good documentation

Borland

For their Delphi RAD tool - just the greatest

Dejan Crnila

For his comms port component for Delphi, v2.63. This is available from Delphi City:

<http://www.delphicity.net>

Microsoft

For their graphical user interface and 32 bit operating system

I would also like to thank all the users of my previous program, JBug11, for their interest and comments, many of which have had a positive influence on Xplor8.

Manual written in:

PDF document editing

Graphics edited in:

Hardware line diagram drawn in:

HC908 assembly language programs written with:

Programs compiled with:

Help file composed with:

Installation program:

WordPerfect 10

Acrobat 5

Paint Shop Pro 7

Isis 5.1 Lite (part of the Proteus suite)

UltraEdit 14 by Ian D Mead

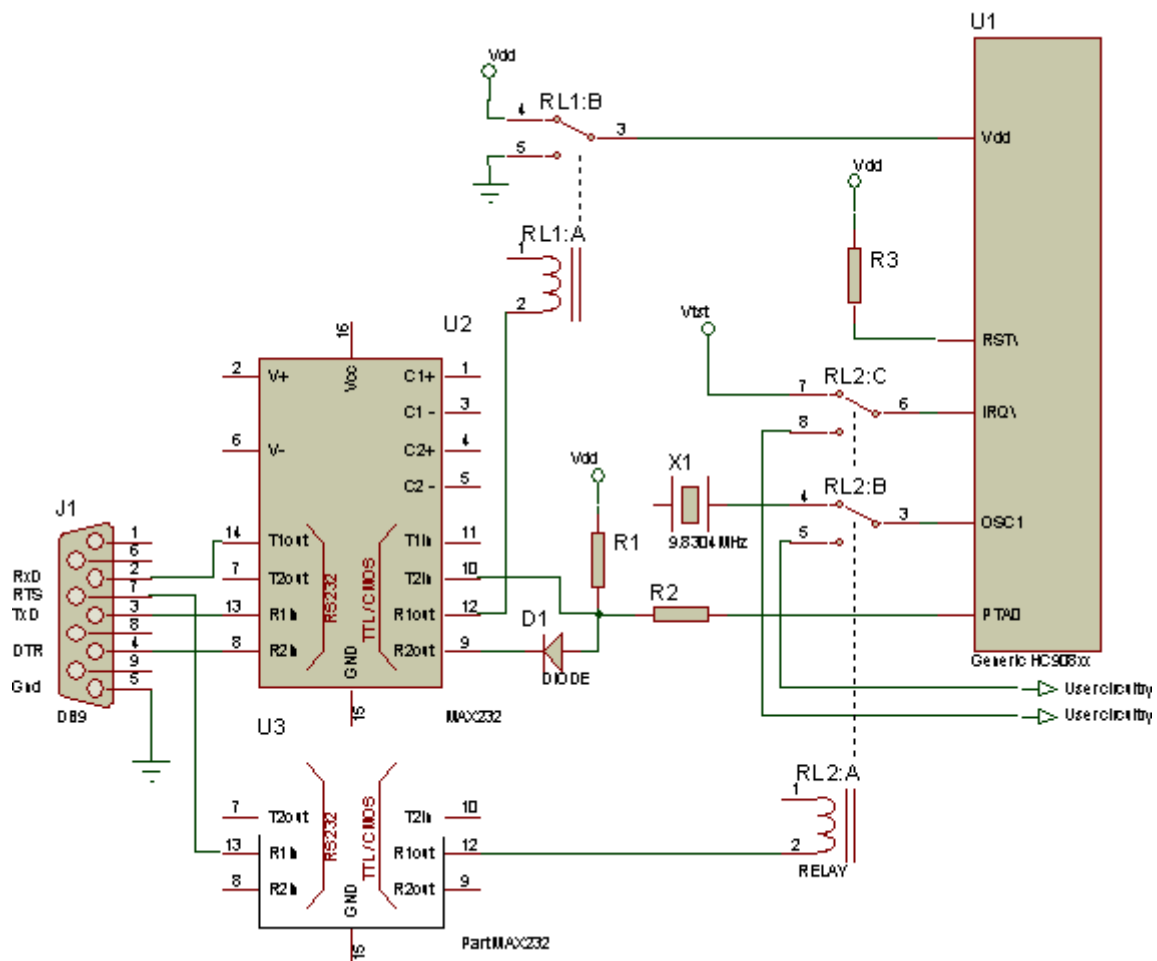
Asm8 by Tony Papadimitriou of Aspisys

HelpHikes Pro by Sanjay Kanade

SIBuilder from PJSoft by Peter Johnson

APPENDIX B - HARDWARE

This diagram shows the basic set-up needed for Xplor8 to communicate with the MCU, to allow Xplor8 to reset the MCU remotely, and possibly make use of the mode switch output.



NOTE: NOT TO BE USED FOR CONSTRUCTION. This diagram is a notional schematic only; intended to illustrate the intent of the various components. Please see the circuit diagrams and pin and voltage information published in the Freescale data sheet for the particular MCU that you are using. I am also not suggesting that electro-mechanical relays are actually used to realize the circuit - once again they are there to indicate an intent.

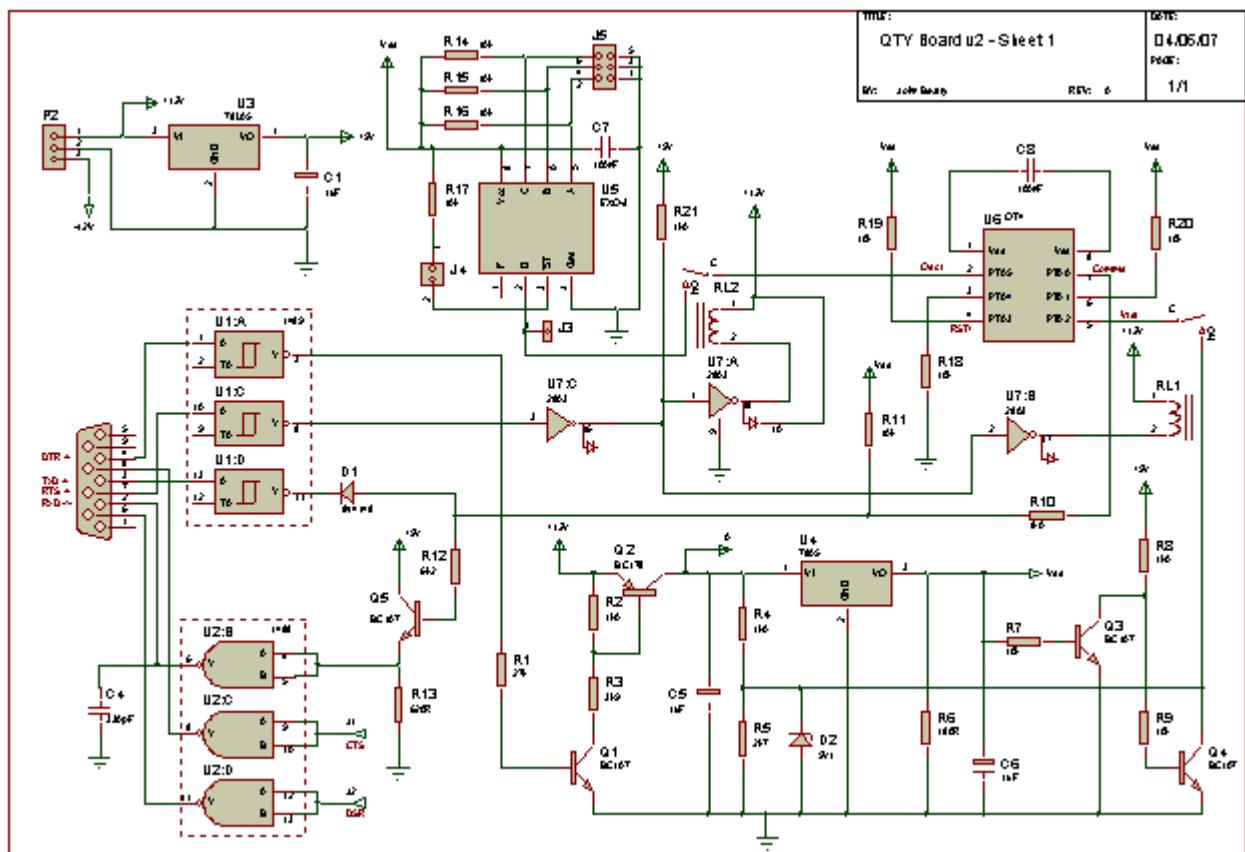
Notes:

1. The most basic setup is PTA0 connected via the simple duplexing circuit of resistors R1, R2 and diode D1 to a level-shifting chip such as a MAX232. See any HC908 data sheet, and Freescale Application Note AN2317: "Low-Cost Programming and Debugging Options for M68HC08 MCUs"
2. None of the other connections needed for a fully operational MCU are shown - any development board should provide the necessary supporting circuitry. In particular certain port pins will need to be pulled up/down at reset - see the data sheet for the MCU that you are using.

3. Relay RL1 performs the power-on reset. A relay is likely to be particularly inappropriate here because of contact bounce.
4. Relay RL2 provides the switching between monitor mode and user mode. This is only required if you wish to switch modes remotely.
5. V_{tst} is the higher-than- V_{dd} voltage needed to access monitor mode. See the individual data sheets for its value.

Xplor8 Development Board

For interest, I attach below the circuit of the board that I used for most of the development work on Xplor8.



APPENDIX C - COMMAND SUMMARY

CLM	Clear the local, Xplor8, copy of memory
CLS	Clear output window
CONNECT / DISCONNECT	Open or close the COM port
CRC Path&FileName.rec s19 CRCL Path&FileName.rec s19 CRC StartAddr EndAddr +Length CRCL StartAddr EndAddr +Length	Compute a CRC-16 sum for a file or a block of MCU memory
EFB Address NewByte	Change a single byte in FLASH memory
F StartAddr EndAddr +Length ByteString CharString	Fill MCU memory with bytes or a character string
FL StartAddr EndAddr +Length ByteString CharacterString	Fill local memory with bytes or a character string
FIND StartAddr EndAddr +Length ByteString CharacterString	Find a byte or character string in MCU controlled memory
FINDL StartAddr EndAddr +Length ByteString CharacterString	Find a byte or character string in local memory
G [StartAddr * [Breakpoint]]	Go - run a program, stopping at a breakpoint if requested.
LD Path&Filename.rec .s19 LD Path&Filename.obj .bin StartAddr	Load MCU memory with an S19 format file, or with a binary image file.
LDL Path&Filename.rec .s19 LDL Path&Filename.obj .bin StartAddr	Load Xplor8 local memory with an S19 format file, or with a binary image file.
L [StartAddr * [EndAddr +Length]]	List MCU controlled memory
LL [StartAddr * [EndAddr +Length]]	List Xplor8 local memory
LM	List macro names
ME [FFFF]	Mass erase FLASH memory
M StartAddr or MM StartAddr	Modify MCU controlled memory
NEXT	Search for further occurrences of string nominated in the FIND command
O[Repeats] [StartAddr *]	Trace over subroutines
PAUSE [milliseconds]	Pause macro execution
PE StartAddr [EndAddr]	Erase one or more pages of FLASH memory

R IOReg CtrlReg [=NewValue]	Display and modify registers
RESET	Remotely reset the target MCU
SV StartAddr EndAddr +Length Path&Filename.Ext <i>or</i> SVL StartAddr EndAddr +Length Path&Filename.Ext	Save a block of MCU or local memory to an S19 or binary image file
SEND BaudRate InterCharDelay ByteString	Send a string of bytes to the MCU
T[Repeats] [StartAddr *]	Trace a program in MCU controlled memory
TERM	Open the terminal window
U[Repeats] [StartAddr *] <i>or</i> U StartAddr * EndAddr +Length	Un-assemble (disassemble) a program in MCU controlled memory
UL[Repeats] [StartAddr *] <i>or</i> UL StartAddr * EndAddr +Length	Un-assemble a program in local memory
UNLOCK [ByteString]	send the security byte string
V Path&Filename.rec .s19 <i>or</i> VL Path&Filename.rec .s19	Verify a program in MCU or local memory against an S19 format file
VE [StartAddr [EndAddr +Length]]	Verify that memory is erased (all bytes \$FF)

APPENDIX D - SETUP FOR DIFFERENT MCU's

Using Xplor8 with different members of the HC908 family.

Utility Files

These are in S19 format and vary with MCU according to the amount of RAM available, and the address of the on-chip auxiliary routines, if any. The source files are also included in the distribution.

Chip Type	Utility files and Function	
HC908GP32	GP32_UT1.S19 GP32_UT2.S19 GP32_UT3.S19	Verifying FLASH programming Page and mass erasing
HC908JB8	JB8_UT1.S19	All functions
HC908- JK1,JK3,JL3	JKL_UT1.S19 JKL_UT2.S19	Verifying and FLASH programming Page and mass erasing
HC908- QT1,QT2,QT4, QY1,QY2,QY4	QTY_UT1.S19 QTY_UT2.S19	Verifying and FLASH programming Page and mass erasing

Register information files

These are in comma-separated value (csv) format.

Chip Type	Register Information File
HC908GP32	Regs_908GP32.csv
HC908JB8	Regs_908JB8.csv
HC908- JK1,JK3,JL3	Regs_908JL3.csv
HC908QT1	Regs_QT1.csv
HC908QT2 & QT4	Regs_QT24.csv
HC908QY1	Regs_QY1.csv
HC908QY2 & QY4	Regs_QY24.csv

APPENDIX E - COMMAND-LINE ERROR MESSAGE SUMMARY

Command Line Errors

There are numerous ways in which what is typed on the command line may not be appropriate. If Xplor8 detects an error, it replies by echoing the command line to the Command history window with a left-pointing arrow and an explanatory message. The following table lists all the errors that may occur. Most are self-explanatory; brief guidance notes are included where appropriate:

Command line error message	Notes
Unrecognized command	Basic error message
Project error - command not available	Project file has errors. Open 'Settings', then click 'OK' for a list of the errors
Not available when disconnected	COM port is closed
Requires Monitor Mode	MCU needs to be running the Monitor firmware. Reset the MCU with the necessary voltages on its pins
Not available in Monitor Mode	
Cannot interrupt running program	Command cannot be used while a program is being run on the MCU
Cannot interrupt while tracing	Command cannot be used while a program is being traced on the MCU
Not available in User mode	
Already disconnected	
Already connected	
Monitor/User switch not available	The General Purpose switching function has not been set up
Already in connected/monitor mode	
Already in user mode	
Address(es) overlap undefined memory	Command arguments, or the loading addresses in an S19 file, overlap undefined memory
Address(es) overlap RAM	(should never occur)
Address(es) overlap I/O Registers	Command arguments, or the loading addresses in an S19 file, overlap the I/O registers
Address(es) overlap ROM	Command arguments, or the loading addresses in an S19 file, overlap undefined ROM

Address(es) overlap FLASH	Command arguments, or the loading addresses in an S19 file, overlap FLASH. Some commands cannot write to FLASH
Address(es) overlap EEPROM	Command arguments, or the loading addresses in an S19 file, overlap EEPROM (where available)
Address(es) overlap reserved memory	
Address(es) overlap locked FLASH	Command arguments, or the loading addresses in an S19 file, overlap FLASH memory when this has not been unlocked
Address(es) protected by FLBPR	You are trying to write to FLASH locations which are protected by the current value of the Flash Block Protect register
Cannot page-erase vector area	Some HC908 chips can only erase the vectors by doing a mass erase
Address span not allowed	Some commands can only write one type of memory at a time
Integer argument expected	
Must follow FIND or NEXT	The NEXT command must immediately follow a use of the FIND or NEXT command
Previous FIND failed	NEXT cannot be used if a previous FIND or NEXT was unsuccessful
Register not found	The register name supplied to the Register Display and Change command was not found.
Argument out of range	
Remote reset not available	The Remote Reset function has not been set up
Formatting error in S19 file	For example: invalid checksum
Argument(s) not recognized	
Too few arguments	
Too many arguments	
Address range error	Usually because EndAddress is less than StartAddress
String too short/long	
Command not available in macros	
Command only available in macros	

BEGIN not found	The 'BEGIN' directive was not found in a macro invoked from the command line
No library macros found	
File not found	
Unrecognized file type	Files for loading or saving can only have the extensions: .rec, .s19, .bin, or .obj
Binary file is too large	The number of bytes in the file exceeds the available memory on the MCU
Invalid opcode at proposed breakpoint	
Invalid opcode at start address	Running or tracing cannot be begun at: (all values in hexadecimal) <ul style="list-style-type: none"> • illegal opcodes (32, 3E, 9E62, 9E65, 9E6E, 82, 8D, 96, AC, 9EDC, 9EDD, 9EEC, 9EED) • indexed jump instructions (DC, EC, FC) • the SWI instruction (83) • BIL or BIH instructions (2E, 2F) • any instruction which jumps or branches to itself
Cannot set breakpoint	A breakpoint cannot be set as for 'Invalid opcode at start address' - see description in box above

APPENDIX F - RS232 COMMUNICATIONS

Introduction

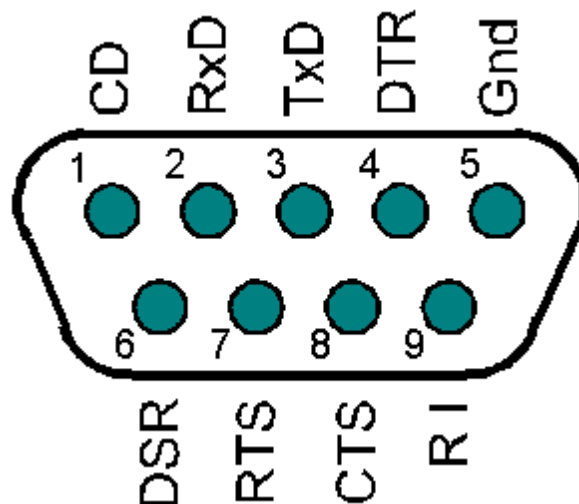
The RS232 standard, now more correctly the EIA232 standard, first appeared in the 1960's. It was originally written to standardize the interconnection of terminals and modems, in the days when one mainframe computer hosted many remote terminals. In the standard, a terminal is referred to as a DTE (*Data Terminal/Terminating Equipment*), and the modem as a DCE (*Data Communication Equipment*). This primer is a brief introduction to the standard, and to the signaling protocol usually employed, from the perspective of a Xplor8 user who just wants to 'get it all to work'

Physical Connectors

The DTE is traditionally fitted with a male 'D' connector, and the modem (DCE) with a female one. When the IBM personal computer first appeared it had serial or 'COM' ports which were implemented as RS232 standard 25-pin male 'D' connectors on the back panel. When the AT version of the PC appeared, IBM began to adopt a nine pin version of the 'D' connector, and this is now universal on personal computers that still have hardware serial ports.

One pin is reserved as the common signal return or ground pin (5). Two pins are used for full duplex data transfer (2,3). The remaining six pins all carry handshaking information to control the flow of data on Tx/D and Rx/D.

The pinout, looking on the outside of the male DB9 connector as found on the back of a PC, is as follows:



The pins are labeled with a functional description as follows. 'IN' and 'OUT' refer to the signal directions as they apply to the DTE:

Pin Number	Abbreviation	IN or OUT	Description
1	CD (or DCD)	IN	Carrier Detect. Used by a modem (DCE) to tell the DTE that it has detected a carrier signal
2	RxD	IN	Received Data. Data sent by the DTE to the DCE
3	TxD	OUT	Transmitted Data. Data sent by the DTE to the DCE
4	DTR	OUT	Data Terminal Ready. Used by the DTE to tell the DCE that it is operational.
5	GND		Ground. Common signal return.
6	DSR	IN	Data Set Ready. Used by the DCE to tell the DTE that it is operational.
7	RTS	OUT	Request to Send. Used by the DTE to signal the DCE that it may begin sending data.
8	CTS	IN	Clear to Send. Used by the DCE to signal to the DTE that it may begin sending data.
9	RI	IN	Ring Indicator. Used by a modem (DCE) to tell the DTE that an incoming call has been detected.

In the case of Xplor8, the personal computer host is a DTE and the target board is a DCE. As explained in Appendix B, the only pins that must be connected are RxD, TxD and GND. Remote resetting will require a connection to DTR (or RTS) also. No other handshaking signals are needed; if connected, they will be ignored.

Voltages, Impedances and Biasing

RS232 is referred to as an 'unbalanced' signaling system in that all the signals are referred to a common ground conductor. This is inherently more susceptible to crosstalk and common-mode sources of interference than, for example, the balanced system described in RS422. Binary state signaling is used, with one state represented by positive voltage on the signaling pin of anywhere between +3V and +15V with respect to the common ground pin (pin 5); and the other state by a negative voltage between -3V and -15V. Voltages within the transition region from +3V to -3V are meaningless.

The source impedance of a signal driver should be such that accidental cross-connection with another output cannot damage either driver even where one is outputting a positive voltage and the other a negative one.

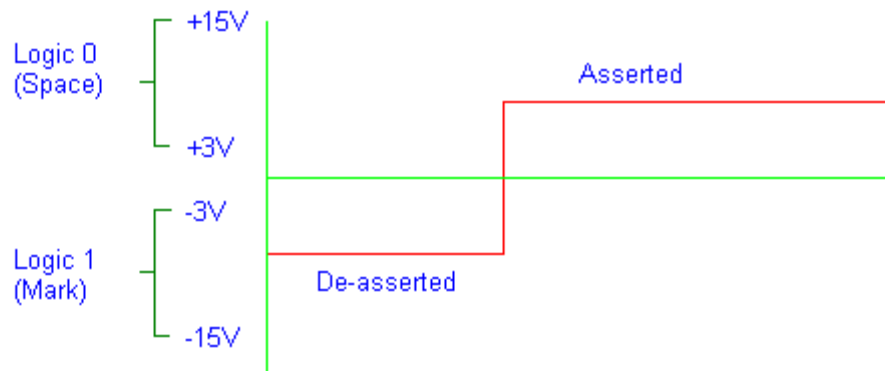
The default state of all signals is de-asserted, i.e. a negative signal voltage. The line receiving devices should be biased so that they 'see' a negative voltage if the cable is disconnected, or if the cable is connected but the transmitting end is unpowered. It may be noted that biasing the RxD pin so that it 'sees' a negative voltage in the absence of a connecting cable, and the fact that a negative voltage is the usual 'idle' line condition, means that there is no automatic indication of a break in the cable.

Logic Levels

It is this aspect which causes the most bewilderment among users, largely because of the counter-intuitive naming conventions used. The negative voltage is referred to as a logic 1 or 'mark' condition and a positive voltage is referred to as logic 0 or 'space' condition.

A driver transmitting data on the TxD pin idles in the logic 1 or 'mark' state, i.e. with a negative voltage output. In the absence of data being actively sent from the remote device, the voltage on the RxD pin will also be negative, because that is the idling state of the driver at the 'far' end of the link.

In the case of the control signals, a de-asserted signal is one in the 'mark' or negative voltage state. For example, DTR is asserted if it goes from the negative (logic 1, mark) state to the positive (logic 0, space) state, as here:

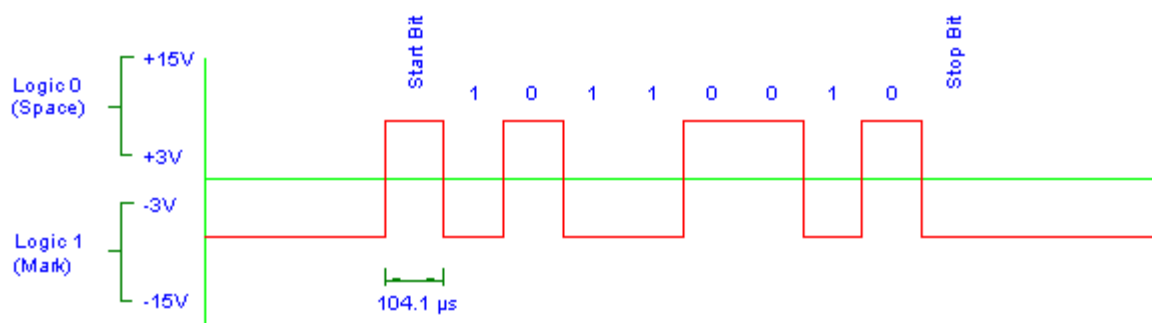


It may be noted that all of the common RS232-to-logic level converters (MAX232, 1488, 1489) invert the polarity of the transmitted and received signals. So a TTL 'high' level, logic 1, +5V, corresponds to a negative RS232 voltage (somewhere between -3V and -15V).

Data Protocols

Serial data on an RS232 link is transmitted asynchronously, i.e. a separate clock signal is not required and the start of a data byte can occur at any time. To achieve this, a logic 0 *start* bit is sent, followed by the data bits, and ending with a logic 1 *stop* bit. I will only consider here the commonest 8N1 format - a start bit followed by **Eight** data bits, **No** parity bit and **One** stop bit. The data bits are sent in least-significant-bit first order. All the bits (start, data and stop) have a duration equal to the reciprocal of the *baud rate*. Note that the technical definition of baud rate is more complex, and it only happens that baud rate and bit rate are equal for the simple binary signaling system used on RS232 links. To send a single byte takes 10 bit times (1.042 ms at 9600 baud). Since a start bit can immediately follow a stop bit, at 9600 baud the maximum throughput is 960 bytes per second.

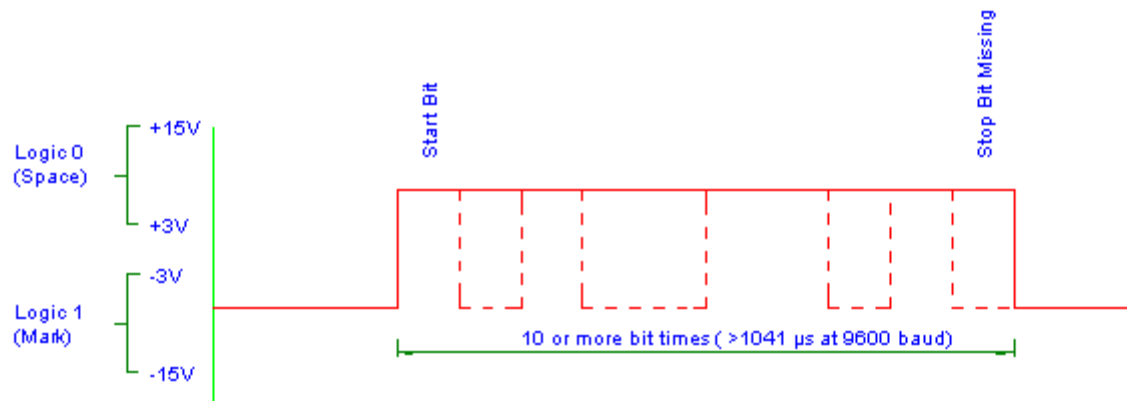
Here is the letter 'M' in ASCII coding (hex 4D, binary 01001101) sent according to the 8N1 format at 9600 baud, as it appears on the TxD pin:



Break Signal

The RS232 standard also specifies a signal called a Break, which is the sending of continuous Space (logic 0) values (no Start or Stop bits). I believe the name dates from the days when many news teleprinters used to be connected in series, and the break signal could be used to alert all machines that an important announcement was pending. As noted above, it does *not* mean a physical break in the connecting cable.

The Break signal must be of a duration longer than the time it takes to send a complete byte plus the Start and Stop (and Parity, if any) bits. The minimum signal which will be considered a break at 9600 baud in 8-N-1 format is shown below:



Not all UARTs will reliably detect a break signal of the minimum length. This applies particularly to [USB-to-Serial adapters](#).

APPENDIX G - GETTING STARTED and QUICK TOUR

Installation

Double click on the self-installing exe file, named something like 'Install-Xplor8-xyz.exe' which you have downloaded from the web site. This is a console application, so will start in a DOS box. You will be given the chance to change the default installation folder (C:\Program files\Xplor8) if you wish to install it somewhere else.

The installation utility should place an icon on the desktop; if it does not, you can right-click on Xplor8.exe and use the context menu to Send To>Desktop (create shortcut).

Launch

Launch Xplor8. If this is the first time that the program is run, you will receive an error message, which may be ignored at this stage (answer OK). If Xplor8 has been previously installed, and a newer release of the program is being started for the first time, a dialog will appear to allow the smooth transfer of configuration information from the previous version.

Adjusting the Displayed Form

As supplied, the main form of the program has quite a small size, and you may wish to increase it using one of the predetermined layouts available under the View menu item.

Connecting the Target Board

Connect the target MCU target board to one of the PC serial ports. The minimum hardware configuration is shown in [Appendix B - Hardware](#). A remote reset capability is a luxury, and not necessary to get up and running, although some form of on-board reset button will still be required. If you use a USB-to-serial adaptor, then read [USB-to-Serial Adapters](#) on page 12 as not all such adaptors are suitable.

Configuring Xplor8

- On the main menu, click 'File' and then 'New Project...'. The Settings dialog will open at the 'General' tab.
- Select the MCU which are using from the drop-down list. This automatically updates the memory map and other chip-specific information that Xplor8 needs.
- Select, or enter, the E-clock frequency at which the MCU is operating.
- Move to the 'COM Port' tab in Settings. Select the COM port to which you connected your target board.
- (Unlikely) If you are using a non-standard crystal frequency, adjust the communication baud rate.
- If your [Hardware](#) allows remote resetting, tick the checkbox 'PC controls reset' and select the control pin to use and the pulse polarity.

- Move to the 'Security Bytes' tab. Enter the sequence of bytes needed to unlock FLASH memory on the MCU. If you are using a brand new chip these will all be FF.
- Close the Settings dialog by clicking 'OK'.

Connecting to the Monitor Firmware (MON08)

- Click the 'Connect' speedbutton at the left hand end of the row of buttons.
- Reboot the MCU. Either using the reset button on the target board, or, if the appropriate hardware is in place, by clicking the 'Reboot Talker' speedbutton.
- Send the eight security bytes, using the 'Unlock' command
- Xplor8 should now receive a 'break' signal from the MCU. If it does so, the status line should read 'Monitor Mode' and, if the security bytes were correct, 'FLASH Unlocked'. If a 'Comms Error' dialog appears, then there is a communication problem. If this persists, consider sending an [Error Report](#), see page 79.

Using Commands

The monitoring and de-bugging functions of Xplor8 are invoked by typing in [Commands](#) into the [Command Edit Box](#), see page 20. Some important commands, such as those to load a file to the MCU, are also duplicated in the Actions menu. See the Quick Tour below for some initial commands to try.

Beginner's Advice

If you are quite new to programming for the HC908 series, I cannot do better than repeat the advice given in the Starter Package manual for the Technological Arts micro controller boards: '..start with something that works, and then add new features incrementally'. The very simple 'Looptest' program in the \Samples\ folder is a possible starting point.

Happy Debugging!

See the next page for a quick tour of Xplor8.

QUICK TOUR

A quick introduction to the capabilities of Xplor8.

The tour assumes that you are able to unlock FLASH memory on the MCU you are using - either because the chip is already fully erased or because you know the security byte sequence. If you are not able to unlock the chip, you will have to mass erase it first - see the last item below.

1. Listing Memory

If you are able to list (or dump) memory contents then at least you know that the PC is interfacing correctly with the monitor firmware on the MCU. Type the following example of the [List Memory](#) command (page 50) on the command line, and press <Enter>:

```
L 0 <enter>
```

Note the space between the 'L' and the zero. This should produce a listing of the first sixteen addresses on the MCU. Why sixteen? - because this is the default for the number of locations to list. It may be changed on the [Settings>General](#) tab. If you type:

```
L 0 2 <enter>
```

then you will get a listing of the first three locations only. Note that address arguments supplied to Xplor8 commands always specify an inclusive range.

2. Reading a control register

Try using the [Register Display and Change](#) command (page 55). Type the following on the command line, but note that some MCUs have a CONFIG1 and CONFIG2 register, so you may have to change the name to suit:

```
R CONFIG <enter>
```

This should produce something like:

```
001F  CONFIG  [$00, # 0] 0 0 URSTD LVID SSREC COPRS STOP COPD
```

Note that the individual bits are named (where appropriate) and displayed in bold if set.

3. Loading an S19 file

Try loading an S19 file to FLASH memory. The 'Samples' sub-folder of the Xplor8 installation folder contains a file named 'Looptest.asm' which has been assembled to start at \$FB00 in the file LOOPTEST.S19. If you have free FLASH at this address (press Ctrl+M for the memory map), then you can proceed to load it immediately, otherwise it will require re-assembling at a suitable address. The files LOOPTEST.LST and LOOPTEST.SYM are also included in the folder. To load the file, click the 'Load S19 to MCU' speedbutton, and navigate to the 'Samples' sub-folder. Open the file LOOPTEST.S19. The output window should display the messages:

```
LD "[Xplor8]\Samples\LOOPTEST.S19"  
Writing  
    FB00..FB23  
Loading complete
```

4. Unassembling Memory

Confirm that you have loaded the 'Looptest' file satisfactorily by unassembling it. Type (upper or lower case):

```
U FB00 FB1E <enter>
```

Compare the listing in the output window with that in the LOOPTEST.LST file which is also in the \Samples\ subfolder.

5. Unassembling with Labels

Open [Settings>Debug](#), and check 'As last loaded S19 file' in the Symbol Table panel. Close Settings by clicking the 'OK' button. Recall the previous unasassembly command by hitting the up arrow once. Hit <enter>. Now the listing should include the symbolic label information for 'Looptest'.

You may view the currently-loaded symbol table by clicking 'Symbol Table' in the View menu. The symbol table display may be adjusted for height, and moved to some other part of the screen - its layout will be remembered in the Windows Registry for the next time it is used.

6. Verifying Memory

You can also verify a loaded program by using the [Verify](#) command (page 66). Click on Actions>Verify S19... (in the main menu), and select LOOPTEST.S19 again in the file-open dialog. This should produce the message 'MCU memory verifies OK' in the output window.

7. Running a Program

Type:

```
G FB00 <enter>
```

The status line will change briefly to 'Running' and then back to 'Monitor Mode'. The LOOPTEST program ends with an SWI which is why it returns to monitor mode. Depending on the selections in [Settings>Debug](#) the output window will display information on the instruction and registers at the starting point, \$FB00.

8. Tracing a Program

Type:

```
T FB00 <enter>
```

Depending on the selections in Settings>Debug, the output window will display information on the instruction and registers at the starting point, \$FB00, and at the break point (\$FB01). The status line should show 'Monitor Mode'. The [Start/Break Point](#) box, part of the information sidebar on the main form, shows in green an abbreviated form of the instruction at the point where you began tracing, with similar information for the breakpoint highlighted in red because the program has halted there. The 'BR' address panel shows the current value in the MCU BRK register. The 'LU' (List & Unassemble) address panel shows the default starting address for list or unassemble operations - it is updated during tracing to reflect the program breakpoint. Note also that the command edit box is pre-filled with 'T' so that simply pressing <enter> will trace the next instruction.

9. Watching a Variable

Add an address to monitor during tracing. In the LOOPTEST program, the counter is stored in 1,SP so: right-click in the Watch Window (main form, lower right-hand corner) and select 'Add...' The [Add to Watch](#) dialog opens, and in the 'Stack Relative' box, choose 1,SP by selecting it from the combo box. Click 'Add & Close'. Now each time you repeat the trace command, the current value of 1,SP is updated in the watch window.

10. Erasing

Try erasing the page containing the Looptest program using the Page Erase command. Type:

```
PE FB00 FB23 <enter>
```

Xplor8 will reply with a short dialog to the effect that the page you are about to erase is larger than the range you typed to the command. This is because pages are only erased in minimum-sized chunks. Answer OK to this dialog and Xplor8 should zap your program. You can check by using the [Verify Erase](#) command (page 67):

```
VE FB00 FB23 <enter>
```

11. Mass Erase

This operation is always possible whether or not the correct security byte sequence is sent. Type:

```
ME <enter>
```

and answer 'OK' at the confirmatory dialog. This should erase all of FLASH; once again, you can check with the Verify Erase command. Typing

```
VE <enter>
```

alone should be sufficient.

***** This concludes the quick tour. *****

APPENDIX H - DISTRIBUTION FILES

The installation file Install-Xplor8-xyz.exe sets up the following files, in the sub-directories listed below. Note that the Xplor8 printed Manual itself is an Adobe Acrobat (.pdf) file which must be downloaded from the web site separately.

Primary installation folder:

ReadMe.txt	General set-up instructions and late-breaking information (not always included)
Xplor8.exe	The main monitor/debugging program
InstallLib.dll	Files connected with installing and uninstalling
UnInstall.exe	

Sub-folder ..\Help\

Xplor8.HLP	On-line help files.
Xplor8.CNT	

Sub-folder ..\MCU\

908GP32.cfg	Configuration information for MCU's
908JB8.cfg	
908JK1.cfg	
908JK3.cfg	
908JL3.cfg	
908QT1.cfg	
908QT2.cfg	
908QT4.cfg	
908QY1.cfg	
908QY2.cfg	
908QY4.cfg	

Sub-folder ..\Opcodes\

HC08_Opcodes.csv	Opcode and instruction information. Applies to all chips
------------------	--

Sub-folder ..\Utility\

GP32_UT1.S19	See Appendix D for an explanation of what each utility does
GP32_UT2.S19	
GP32_UT3.S19	
JB8_UT1.S19	
JKL_UT1.S19	
JKL_UT2.S19	
QTY_UT1.S19	
QTY_UT2.S19	
GP32_UT1.ASM	Source files for the above
GP32_UT2.ASM	

GP32_UT3.ASM
 JB8_UT1.ASM
 JKL_UT1.ASM
 JKL_UT2.ASM
 QTY_UT1.ASM
 QTY_UT2.ASM

Sub-folder ..\Projects Default location for project files.

Sub-folder ..\Registers

Regs_908GP32.csv	Control register information for the 908GP32
Regs_908JB8.csv	Control register information for the 908JB8
Regs_908JL3.csv	Control register information for the 908JL3
Regs_908QT1.csv	Control register information for the QT and QY chips
Regs_908QT24.csv	
Regs_908QY1.csv	
Regs_908QY24.csv	

Sub-folder ..\Samples

TermDemo.asm	HC908 assembly language program to demonstrate the use of the terminal window. See Terminal Demonstration Program.
Looptest.asm	Elementary HC908 program to demonstrate the tracing of branching instructions
LOOPTEST.S19	Above program assembled to start at \$FB00, in Motorola S19 format. Assembler output listing Symbol table for above program
LOOPTEST.LST	
LOOPTEST.SYM	
OddBrk.asm	Assembly language program which, when assembled and run, shows up the odd behaviour of the HC908 Break Module in certain circumstances.
OscTrim.asm	Program to trim the internal oscillator available on certain HC908 chips. See this source code file for an explanation of its use.
JB8.hdr	Resource header files
QY4.hdr	
String.rtn	String handling routines
Sample1.mac	Sample macro library file

APPENDIX J - Xplor8 REVISION HISTORY & KNOWN BUGS

Revision History

Known Bugs: Does not display correctly on operating systems, such as Microsoft Vista®, when using displays set to 120 dpi or more. Display is effectively unusable. Only current workaround is to use 96 dpi.